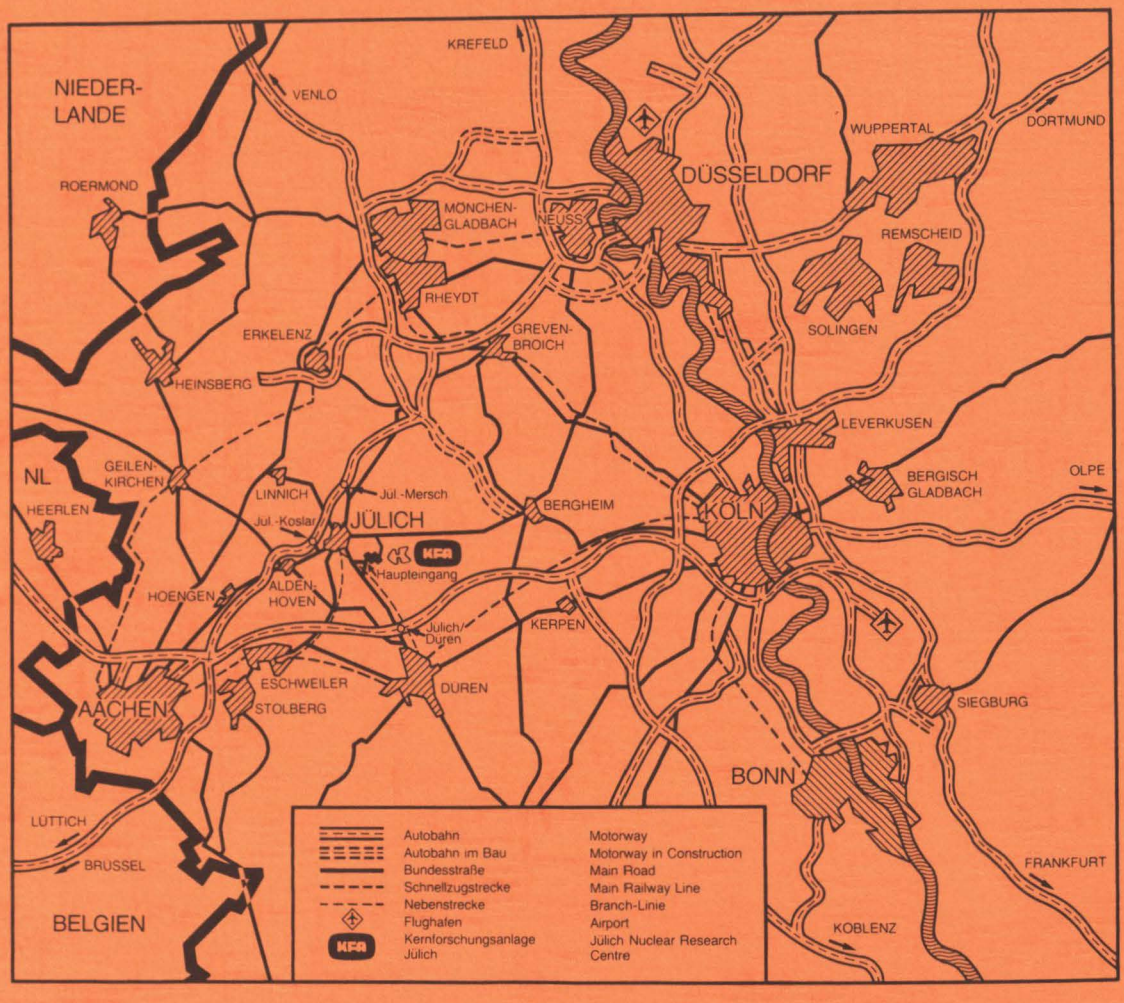
KERNFORSCHUNGSANLAGE JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik

Numerische Lösung von Warteschlangennetzwerken auf Pipelinerechnern

von
R.-D. Rumler

Jül - Spez - 218
September 1983
ISSN 0343-7639



Als Manuskript gedruckt

Spezielle Berichte der Kernforschungsanlage Jülich – Nr. 218
Zentralinstitut für Angewandte Mathematik Jül - Spez - 218

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH
Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)
Telefon: 02461/610 · Telex: 833556-0 kf d

Numerische Lösung von Warteschlangennetzwerken auf Pipelinerechnern

von
R.-D. Rumler

Diese Arbeit wurde als Diplomarbeit am Institut für allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen bei Herrn Prof. Dr.-Ing. O. Lange im Zentralinstitut für Angewandte Mathematik der KFA Jülich angefertigt.

Meinen Dank für die Unterstützung durch das Zentralinstitut für Angewandte Mathematik richte ich an den Direktor des Instituts, Herrn Dr. F. Hoßfeld.

INHALTSVERZEICHNIS

1.0 EINFÜHRENDER ÜBERBLICK

Ein Rechensystem bearbeitet Aufträge. Eine Beurteilung der Art und Weise und der Effizienz ist auf verschiedenen Wegen möglich. Als Hilfsmittel wird hier ein Warteschlangenmodell des Rechensystems herangezogen. Die Vorgänge in diesem Modell, von denen angenommen wird, daß sie solche des realen Systems annähern, sind durch Übergangswahrscheinlichkeiten bzw. -raten von einem Zustand des Modells in einen möglichen Folgezustand beschreibbar. Bei sinnvoller Anordnung dieser Raten oder Wahrscheinlichkeiten in Form einer Matrix sind zur Bestimmung eines Gleichgewichtszustands des Modellsystems numerische Methoden anwendbar. Die Warteschlangenlängen, durchschnittliche Verweildauer eines Auftrags im System etc. können dann leicht berechnet werden.

Die Gestaltung des Modells und die Auswahl der numerischen Verfahren erfolgte anhand der Dissertation von B. Müller (MUE 80). Teile dieser Verfahren scheinen für einen Pipelinerechner geeignet zu sein. Weil Warteschlangennetzwerke der hier betrachteten Form zwar einen endlichen, aber im allgemeinen sehr großen Zustandsraum haben, verspricht die Verwendbarkeit eines Pipelinerechners eine wesentliche Verkürzung der Rechenzeit auch für Algorithmen zur Lösung solcher Probleme.

Im Kapitel 2 werden die Grundlagen des Pipelining skizziert, ein Überblick über Aufbau und Funktionsweise des verwendeten Rechners gegeben und die verwendete Hilfssoftware genannt.

Das 3. Kapitel beschäftigt sich mit den mathematischen Grundlagen der verwendeten Verfahren aus der Wahrscheinlichkeits- und Warteschlangentheorie sowie, im Anschluß, mit der Schilderung der numerischen Verfahren selbst.

Eine Klasse von Warteschlangennetzwerkmodellen wird im 4. Kapitel ausführlich beschrieben. Dabei wird die Verbindung zu den im vorigen Kapitel aufgeführten Methoden hergestellt und die für diese Verfahren wesentlichen Eigenschaften der Modellklasse genannt. Alle vorgenommenen Vereinfachungen des Modells sind erwähnt.

Es schließt sich das Kapitel über die Algorithmen selbst an. Dort findet man im wesentlichen einen Überblick über die benutzten Datenstrukturen und die Einzelbeschreibung der Programme.

Den letzten Punkt bildet eine qualitative Schilderung der Testläufe unter Skizzierung der jeweils dafür notwendigen Änderungen.

2.0 HARDWAREVORAUSSETZUNGEN UND HILFSSOFTWARE

2.1 PIPELINERECHNER

2.1.1 DAS LINEARE PIPELINEKONZEPT

Ein wesentlicher Engpaß bei der Verarbeitung von Instruktionen und Daten in einem nach dem von-Neumann-Prinzip gebauten Rechner liegt in deren sequentieller Abarbeitung. Es gibt verschiedene Wege, um einen parallelen Ablauf zu erreichen.

Eine Möglichkeit bietet das Pipelinekonzept. Einführend wird das lineare Pipelining beschrieben. Dabei wird ein sequentiell ablaufender Verarbeitungsschritt geeignet in k je wiederum sequentielle, aber voneinander nahezu unabhängige Teilprozesse zerlegt.[1] Jeder Schritt hängt nur insofern vom vorhergehenden ab, als dessen Resultat für ihn die Eingabe darstellt. Die Hardware-Einheiten (**Pipelinesegmente**), welche diese Prozesse realisieren, können parallel arbeiten, so daß gleichzeitig k Teilschritte von k der ursprünglichen Operationen ablaufen können.

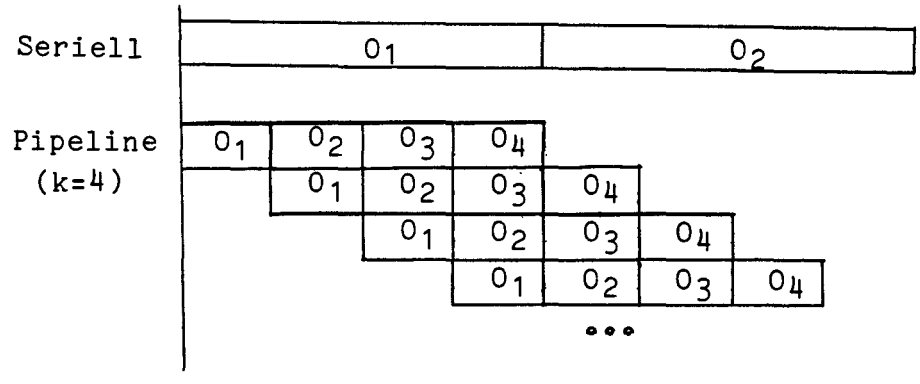


Bild 2.1.1: Segmentierung einer Operation

Pro Segment ist zur Synchronisation der Pipeline ein Zwischenspeicher (latch) zur Aufnahme des jeweiligen Ergebnisses notwendig.

Man unterscheidet zwei Arten des Pipelining (RAL 77):

- Instruktionspipelining

  Die Verarbeitung einer Instruktion in der Instruktionseinheit eines Rechners wird üblicherweise in vier Teilschritte zergliedert:
  1. Instruktion holen
  2. Instruktion dekodieren
  3. Operanden holen
  4. Operation ausführen

  Wird pro Funktion ein Pipelinesegment mit der

---

[1] Die im Abschnitt 2.0 verwendeten Symbole sind unabhängig von denen im übrigen Teil dieser Arbeit.

Verarbeitungszeit Δt bereitgestellt, so ist nach der **Aufsetzzeit** 3 Δt, die zum Füllen der Pipeline benötigt wird, pro Δt eine Instruktion beendet.

- Funktionspipelining

  Der wesentliche Unterschied zum Instruktionspipelining liegt in der Art der Prozesse, die segmentiert werden. Es sind hier die Operationen im Rechenwerk, z. B. Addition und Multiplikation von Gleitkommazahlen.

  Für eine Addition ist beispielsweise folgende Gliederung denkbar (SOM 83):
  1. Exponentenvergleich
  2. Mantissenangleich
  3. Mantissenaddition
  4. Normalisierung
  5. Rundung

Der Geschwindigkeitsgewinn (**speedup**) $S_k(n)$ durch Pipelining mit k Segmenten wird definiert als das Verhältnis der Zeit $T_1(n)$ für das Ausführen von n Operationen durch einen sequentiellen Monoprozessor und der dafür benötigten Zeit $T_k(n)$ des Pipelineprozessors.

Falls jedes Segment in der Zeit $\Delta t = T_1(1)/k$ eine Verarbeitung abgeschlossen hat, gilt für $T_k(n)$:

$$T_k(n) = (k - 1)\ \Delta t + n\ \Delta t$$

wobei $(k-1)\Delta t$ die Aufsetzzeit ist. Es ist $T_1(n) = n\ T_1(1)$. Demnach:

$$S_k(n) = T_1(n) / T_k(n) = k\ n / (k - 1 + n)$$

Für großes n ist $S_k(n)$ höchstens k.

2.1.2 DIE RESERVIERUNGSTAFEL

Diese Beschreibung folgt der Darstellung in (SOM 83).

Als Hilfsmittel bei der Beschreibung der Vorgänge in einer Pipeline wird die Reservierungstafel verwendet. Jede Zeile dieser Tafel entspricht einem Pipelinesegment, jede Spalte einer Verarbeitungszeiteinheit.

Ein Eintrag in Zeile i und Spalte j kennzeichnet die Aktivität des Segments i zum Zeitpunkt j. Mehrere Markierungen in einer Zeile zeigen mehrfache Aktivität des entsprechenden Segments zu verschiedenen Zeitpunkten an, mehrere Markierungen in einer Spalte bedeuten paralleles Arbeiten mehrerer Segmente.

Mit Hilfe der Reservierungstafel kann der Ablauf einer Operation in der Pipeline dargestellt werden. Hat man mehrere, nicht unbedingt unterschiedliche Operationen mit ihren Reservierungstafeln, dann kann man damit eine gemeinsame Tafel für eine gemeinsame Ausführung der Operationen bei möglichst hoher Auslastung der Pipeline ermitteln. Die Pipeline ist umso besser ausgelastet, je mehr Segmente zu jedem Zeitpunkt aktiv sind. Man ist daher bestrebt, möglichst viele Operationen zu initiieren. Dabei ist jedoch darauf zu achten, daß zu keinem

Zeitpunkt der Ausführung dasselbe Pipelinesegment von zwei Operationen gleichzeitig benötigt wird, d. h. daß keine **Kollision** auftritt. Eben dies ist anhand der Reservierungstafeln für die beteiligten Operationen feststellbar.

2.1.3 WEITERE PIPELINEKONZEPTE

Bei nichtlinearen Pipelines können pro Segment mehrere Nachfolger auftreten, es kann Rückkopplungen geben (die Ausgabe eines Segments kann an ein in der Ausführung schon durchlaufenes zurückgegeben werden), einige Teiloperationen können parallel auf entsprechenden gleichzeitig aktiven Segmenten durchgeführt werden und für eine Operation kann eine bestimmte Teiloperation mehrfach zu verschiedenen Zeitpunkten im selben Segment bearbeitet werden.

Bei näherer Betrachtung der Funktionsweise von Pipelines sind folgende Klassifizierungen möglich (RAL 77):

* Nach Abfolge der Aktivitäten der Pipelinesegmente:

  - Statische Pipeline.

    Jede Ausführung einer Operation erfordert die gleiche Reihenfolge von Aktivitäten in der Pipeline, d. h. die die Aktivität veranschaulichende Reservierungstafel ist immer dieselbe.

  - Dynamische Pipeline.

    Die Reihenfolge der Aktivität der Pipelinesegmente ist jederzeit veränderlich. Es sind also verschiedene Operationen und demnach unterschiedliche sie beschreibende Reservierungstafeln möglich.

* Nach der Anzahl der ausführbaren unterschiedlichen Operationen:

  - Einfunktionale Pipeline.

    Nur eine Operationsart kann ausgeführt werden. Zwangsläufig ist diese Art der Pipeline statisch.

  - Mehrfunktionale Pipeline.

    Es können unterschiedliche Operationen ausgeführt werden. Ist der Start einer Operation jederzeit möglich, sofern nur die benötigten Pipelinesegmente zum jeweiligen Zeitpunkt frei sind, so ist die Pipeline dynamisch. Falls ein Operationswechsel nur zu bestimmten Zeitpunkten, z. B. nach Ablauf der vorhergehenden Operation, möglich ist, liegt eine statische Pipeline vor.

2.2 DER VERWENDETE PIPELINERECHNER AP-190L

Der für die Zwecke dieser Arbeit eingesetzte Pipelinerechner ist der AP-190L der Firma Floating Point Systems, INC., Portland/Oregon, im folgenden **AP** genannt. Hier soll eine Übersicht über den Aufbau und vorhandene Softwarehilfsmittel gegeben werden.

Detaillierte Beschreibungen finden sich in (FPS 78), (FPS 80).

Der AP kann nicht als eigenständiger Rechner betrieben werden. Der notwendige Steuerrechner (**Host**) muß alle Daten, auch fertigen AP-Maschinenprogrammcode, adreßrichtig übertragen, die Ausführung starten und nach deren durch den AP angezeigtem Ende die Ergebnisse aus diesem lesen.

Im folgenden werden von der Hardware nur die für den internen Ablauf des AP notwendigen Teile beschrieben, nicht die Ein/Ausgabekomponenten. Die Beschreibung folgt der Übersicht in (SOM 83).

2.2.1 AUFBAU DES AP-190L

Im AP sind mehrere Speichereinheiten, Registersätze und einzelne Register vorhanden. Es gibt zwei als Pipeline aufgebaute Gleitkommaeinheiten und eine Einheit zur Berechnung von Programmadressen. Alle Komponenten sind durch verschiedene Datenbusse miteinander verbunden. Das Bild 2.2.1 soll einen Überblick vermitteln.

Die Taktzeit des AP beträgt 167 ns für alle Teile. Man kann drei Datenformate unterscheiden: 38-Bit-Gleitkommaformat (10 Bit Exponent, 28 Bit Mantisse), 16-Bit-Adreßformat und 64-Bit-Instruktionsformat.

Register und Speicher

Im AP sind zwei Datenregistersätze - **"Data pad X"**, **"Data pad Y"** (**DPX,DPY**) -, ein Adreßregistersatz (**S-Pad**), ein Datenspeicher (**MD**), ein Tabellenspeicher (**TM**) und ein Programmspeicher (**PS**) enthalten sowie vier 16-Bit-Register: für die Programm-instruktionsadresse (**PSA**), die Datenspeicheradresse (**MA**), die Tabellenspeicheradresse (**TMA**) und die Datenregisteradresse (**DPA**).

Die beiden Datenregistersätze bestehen aus jeweils 32 Gleitkommaregistern. Sie dienen als Zwischenspeicher der Recheneinheiten oder Datenpuffer zwischen ihnen und dem Datenspeicher.

In jedem Zyklus kann in jeden Registersatz ein Wert geschrieben und gleichzeitig einer gelesen werden, sogar jeweils im selben Register, dann muß aber erst das Lesen erfolgen.

Der Adreßregistersatz (S-Pad) besteht aus 16 16-Bit-Registern, die ganzzahlige Werte zu unterschiedlicher Verwendung aufnehmen können (z. B. Indizierung, Schleifenzählung).

Der Datenspeicher (**Main Data Memory**) hat eine Größe von 64K Gleitkommaworten und ist vierfach verschränkt. Dabei umfaßt Bank 0 alle geradzahligen Adressen bis einschließlich 32766, Bank 1 die entsprechenden ungeradzahligen. Bank 2 und 3 teilen sich analog den übrigen Bereich bis 64535. In jedem zweiten Takt kann auf dieselbe Bank ohne Störung zugegriffen werden, in jedem Takt auf verschiedene Bänke. Nach drei Takten ist der angeforderte Wert im Ausgaberegister (**MD**) oder aus dem Eingaberegister (**MDI**) in den Speicher gebracht.

Der Tabellenspeicher umfaßt 4K 64-Bit Instruktionen. Er wird über das PSA Register adressiert, in welches während der Ausführung einer Instruktion die Adresse der nächsten geladen wird.

Die Adreßeinheit (S-Pad) dient hauptsächlich zur Berechnung von Datenadressen und zur Schleifenzählung. Sie enthält eine 16-Bit arithmetisch-logische Einheit zur Durchführung entsprechender Operationen auf den S-Pad Registern.

Gleitkommaeinheiten

Der AP enthält einen als zweistufige lineare Pipeline gebauten Addierer, der außer der Addition auch noch andere Gleitkommaoperationen durchführen kann, und einen dreistufigen Gleitkommamultiplizierer, der ebenfalls als lineare Pipeline konzipiert ist.

In beiden Pipelines kann pro Takt eine Operation eingeleitet werden.

Der gesamte AP kann als dynamische, mehrfunktionale Pipeline aufgefaßt werden.

2.2.2 HILFSSOFTWARE

Im Rahmen dieser Arbeit wurde nur der für FORTRAN notwendige Teil der vorhandenen Software verwendet. Er besteht aus:

- APFTN

  Dies ist ein in FORTRAN geschriebener FORTRAN Compiler, der aus gegebenem Quelltext ein AP Objekt Programm erzeugt.

- APLOAD

  Die Ausgabe von APFTN stellt die Eingabe für APLOAD dar. APLOAD erzeugt ein AP Lademodul. Außerdem wird für jedes vom Host aus aufgerufene Programm ein Interface-Programm (in FORTRAN) erstellt, das die nötigen Daten- und Programmcodetransfers einleitet, die Ausführung im AP startet und auf deren Ende wartet.

  Die von APLOAD generierten FORTRAN Programme werden zusammen mit den übrigen, im Host verbleibenden Programmen weiter bearbeitet (z. B. Kompilation, Binden).

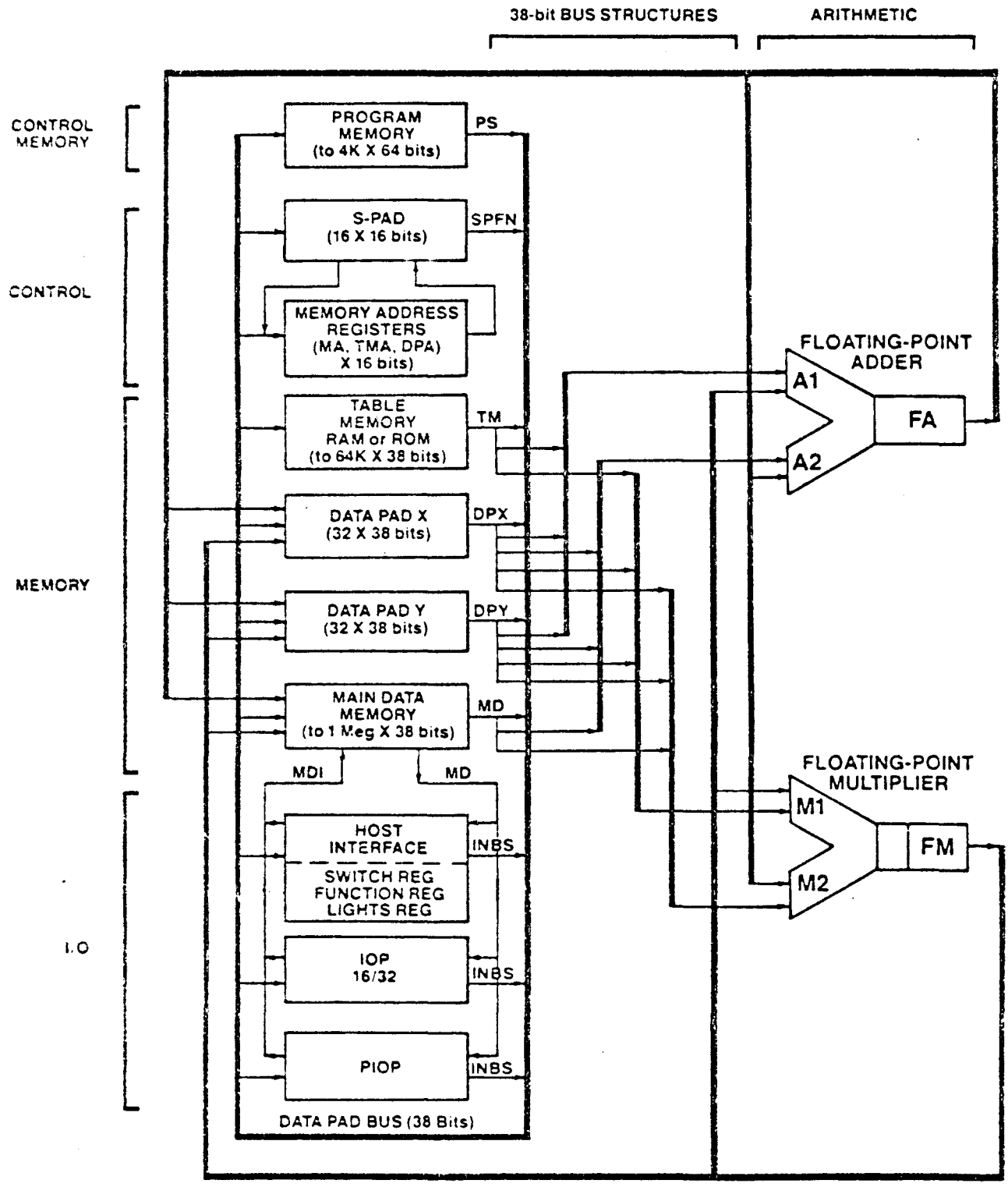
- Bibliotheksroutinen, die auf dem AP laufen.

  FDOTPR zur Berechnung des Skalarprodukts zweier Vektoren,

  FVSMUL zur Multiplikation der Elemente eines Vektors mit einem Skalar,

  FVDIV zur elementweisen Division zweier Vektoren.

Bild 2.2.1: Aufbau des AP-190L



3.0 MATHEMATISCHE VORAUSSETZUNGEN

Schreibweisen:

$\mathbb{N}$ bezeichnet die Menge der natürlichen Zahlen einschließlich der 0. $\mathbb{N}_i$ ist die Menge der natürlichen Zahlen von 0 bis i, je einschließlich. ${}_i\mathbb{N}_j$ umfaßt die Menge der natürlichen Zahlen von i bis j, jeweils einschließlich.

3.1 GRUNDLEGENDE BEGRIFFE

3.1.1 EINFÜHRUNG

Die für diese Arbeit wesentlichen Begriffe aus der Wahrscheinlichkeitstheorie werden hier überblicksartig vorgestellt. Einzelheiten und Zusammenhänge sind z. B. (BAU 78), (CHU 78) zu entnehmen.

3.1.2 WAHRSCHEINLICHKEITSTHEORETISCHE BEGRIFFE

Sei Ω eine nichtleere Menge, die **Menge der Elementarereignisse.** Eine Familie von reellen, meßbaren Funktionen $X(t): \Omega \to \mathbb{R}$ mit Index t, wobei t den Zeitablauf darstellen soll, heißt **stochastischer Prozeß.** Jedes X(t) heißt **Zufallsvariable** über Ω. Falls Ω abzählbar ist, so ist X(t) eine **diskrete Zufallsvariable.** Aus dem **Wahrscheinlichkeitsmaß P** über Ω und der Meßbarkeit von X(t) ergibt sich die **Wahrscheinlichkeitsverteilung F** von X(t) als

$$F(x) := P(\{\omega \in \Omega \mid X(t,\omega) \leq x\}) =: P(X(t) \leq x).$$

(Es werden hier nur Prozesse mit identisch verteilten Zufallsvariablen betrachtet, daher wird der unterscheidende Index t bei der Funktion F weggelassen.)

Falls Ω nicht abzählbar ist, kann man Zufallsvariable X(t) mit **Dichten** f, sogenannte **stetige Zufallsvariable**, definieren:

$$dF(x) = f(x)\,dx \text{ , sofern F differenzierbar ist.}$$

Es gilt also

$$F(x) = \int_{-\infty}^{x} f(u)du = P(X(t) \leq x)$$

Zur vollständigen Beschreibung eines stochastischen Prozesses ist außerdem die Angabe der **gemeinsamen Verteilung $F(\underline{x},\underline{t})$** von m Zufallsvariablen $X(t_1), X(t_2), \ldots, X(t_m)$ mit $\underline{t} = (t_1, t_2, \ldots, t_m)$, $\underline{x} = (x_1, x_2, \ldots, x_m)$ erforderlich. Sie ist gegeben durch

$$F(\underline{x}, \underline{t}) = P(X(t_1) \leq x_1, X(t_2) \leq x_2, \ldots, X(t_m) \leq x_m).$$

Bei Zufallsvariablen mit Dichten $f(\underline{x}, \underline{t})$ gilt der zur einfachen Verteilung analoge Zusammenhang zwischen $F(\underline{x}, \underline{t})$ und $f(\underline{x}, \underline{t})$.

3.1.3 KLASSIFIZIERUNGEN STOCHASTISCHER PROZESSE

Diesem Abschnitt liegt die entsprechende Darstellung in (KLE 75) zugrunde.

Anhand verschiedener Eigenschaften des Wertebereichs von X(t), der Indexmenge aller t und der gemeinsamen Wahrscheinlichkeitsverteilung $F(\underline{x}, \underline{t})$ können stochastische Prozesse eingeteilt werden. Die hier wesentlichen Klassifizierungen folgen, geordnet nach dem Träger der unterscheidenden Eigenschaft.

* Zustandsraum

  Der **Zustandsraum** einer Zufallsvariablen X(t) ist die Menge aller Werte (**Zustände**) dieser Variablen. Ist der Zustandsraum aller Zufallsvariablen des stochastischen Prozesses abzählbar, so heißt der Prozeß **zustandsdiskret** oder auch **Kette**, sonst **zustandskontinuerlich**. Ein **endlicher Prozeß** besitzt einen endlichen Zustandsraum.

  Ein Prozeß ist **irreduzibel**, falls jeder Zustand des Prozesses von jedem anderen erreicht werden kann, sonst **reduzibel**. Erreichbar bedeutet hier, daß die Wahrscheinlichkeit, von einem Zustand zu einem anderen zu gelangen, nicht Null ist. Für die im folgenden definierten endlichen Markovketten ist dieser Begriff analog zum Begriff der Irreduzibilität von Matrizen bzw. zu stark zusammenhängenden Graphen interpretierbar.

* Indexmenge aller t

  Falls die Menge aller Indizes t abzählbar ist, d. h. Zustandswechsel nur zu bestimmten Zeiten möglich sind, bezeichnet man den stochastischen Prozeß als **zeitdiskret**, sonst, d. h. bei Zustandswechseln zu beliebigen Zeitpunkten, als **zeitkontinuierlich**.

* gemeinsame Wahrscheinlichkeitsverteilung

  - Stationärer Prozeß

    Ein stochastischer Prozeß ist **stationär**, wenn seine gemeinsame Verteilungsfunktion bei Zeitverschiebungen unverändert bleibt, also

    $$F(\underline{x}, \underline{t} + \Delta t) = F(\underline{x}, \underline{t})$$

    für alle $\underline{x}$ und $\underline{t} + \Delta t = (t_1 + \Delta t, \ldots, t_n + \Delta t)$.

  - Markovprozeß

    Es werden nur zustandsdiskrete Markovprozesse (Markovketten) betrachtet. Eine Kette heißt Markovkette, wenn die Wahrscheinlichkeit, zum Zeitpunkt t_{n+1} den Zustand $X(t_{n+1})$ vorzufinden, nur vom vorhergehenden Zustand $X(t_n)$ abhängt, formal:

$$P(X(t_{n+1})=x_{n+1} \mid X(t_n)=x_n, X(t_{n-1})=x_{n-1}, \ldots, X(t_1)=x_1)$$

$$= P(X(t_{n+1})=x_{n+1} \mid X(t_n)=x_n)$$

und $t_1<t_2<\ldots<t_{n+1}$.

Bei Markovketten genügt es, als Zustandsraum die Menge der natürlichen Zahlen, je nach Erfordernis beginnend mit 0 oder 1, zu wählen.

Die **Übergangswahrscheinlichkeit** $\hat{p}_{ij}(t_1,t_2)$ von einem Zustand i in einen Zustand j ist dann

$$\hat{p}_{ij}(t_1,t_2) := P(X(t_2)=j \mid X(t_1)=i).$$

Falls $\hat{p}_{ij}(t_1,t_2)$ nicht von den Zeitpunkten t_1 und t_2 abhängt, sondern nur von deren Abstand, also

$$\hat{p}_{ij}(t_1,t_2) = \hat{p}_{ij}(t_1+\Delta t, t_2+\Delta t) =: p_{ij}(t_2-t_1)$$

für alle t_1 und t_2, bezeichnet man die Markovkette als **zeitlich homogen.**[2)]

3.2 WARTESCHLANGENTHEORIE

Im Rahmen dieser Arbeit werden Warteschlangennetzwerke zur Modellierung von Rechensystemen verwendet. Weil mehrere Aufträge in Konkurrenz um die vorhandenen Betriebsmittel treten und dabei i. allg. auf deren Freiwerden warten müssen, ist diese Wahl naheliegend.

Zur Untersuchung des gewählten Modells wird es als stochastischer Prozeß aufgefaßt. Weil die Ankunftzeiten der Aufträge sowie deren Anforderungen - und damit insbesondere deren Abarbeitungszeit - i. allg. nicht genau bekannt sind, ist diese Sichtweise gerechtfertigt.

Alle daher notwendigen Bezeichnungen werden nun eingeführt (MUE 80).

* Warteschlangennetzwerk

  Ein Warteschlangennetzwerk (WS-Netz) besteht aus M Stationen, die durch gerichtete Kanten verbunden sind.

* Station

  Eine Station umfaßt B Bediener und einen Warteraum mit endlicher oder unendlicher Kapazität.

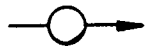
* Bediener

  Ein Bediener benötigt zur Bearbeitung eines Auftrags Zeit. Sie ergibt sich aus den Besonderheiten des Auftrags und des Bedieners und dem Zustand des Systems. Dies wird dahingehend vereinfacht, daß eine wahrscheinlichkeitstheoretische Bedienzeitverteilung als Näherung angenommen wird, deren Parameter vom Systemzustand und der Bedienstrategie des Warteraums der Station abhängig sind.

[2)] Eine zeitlich homogene Markovkette ist demnach eine Markovkette mit stationären Übergangswahrscheinlichkeiten.

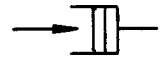
Ein Bediener kann in mehreren Phasen arbeiten. Jeder Phase wird ein Index zugeordnet. Befindet sich ein Bediener in der Phase i, so wird gesagt, er befindet sich im **(aktiven) internen Zustand** i.

Graphisches Symbol pro Phase (s. auch Beispiel 6.1.1):



* Warteraum

Der Warteraum einer Station ist gekennzeichnet durch seine Kapazität und seine Bedienstrategie, die festlegt, in welcher Reihenfolge die wartenden Aufträge dem Bediener zugewiesen werden.

Graphisches Symbol (s. auch Beispiel 6.1.1): 

* Bedienzeitverteilung

Es werden Markovketten zur Darstellung des Geschehens im WS-Netz benutzt. Entsprechend werden nur exponentielle Bedienzeitverteilungen oder daraus zusammengesetzte Verteilungen (COX-Verteilungen) für Bediener mit mehr als einer Phase betrachtet.

* Bedienstrategie

Mögliche Strategien sind z. B.

- FIFO "first in - first out".
- LIFO "last in - first out".
- RR "round robin".
- IS "infinite server". Jeder ankommende Auftrag wird sofort vom Bediener bearbeitet, und zwar so schnell, als wäre nur ein einziger Auftrag vorhanden.
- PS "processor sharing". Alle ankommenden Aufträge werden sofort bearbeitet, aber mit steigender Zahl der Aufträge sinkt die Bedienrate.

Die gerichteten Kanten zwischen den Stationen des WS-Netzes geben die möglichen Wege der Aufträge im Netz an. Dabei hängt die Auswahl des Weges eines Auftrags durchs WS-Netz sowohl von der Art des Auftrags - beschrieben durch seine "Klasse" - als auch von den beteiligten Stationen und deren Zustand ab. Vereinfachend wird angenommen, daß alle Aufträge nur zu einer Klasse gehören.

Die Stationen werden durchnumeriert. Dann kann die Wahrscheinlichkeit des Übergangs eines Auftrags von der Station i zur Station j als w_{ji} bezeichnet werden.

Diese Werte lassen sich in einer Matrix, der **Wechselmatrix W**, anordnen. Um später die Beschreibung der benötigten numerischen Verfahren zur Lösung linearer Gleichungssysteme ausgehend von deren üblicher Form $\underline{A}\underline{x} = \underline{b}$ (und nicht der transponierten Darstellung) vornehmen zu können, wurde die angegebene Indizierung gewählt. Die Besetzungsstruktur der Matrix W liegt aufgrund des jeweiligen Rechensystemmodells fest, dagegen nicht unbedingt die Größe der Werte der Elemente. Diese kann z. B. vom Systemzustand abhängen.

Ein WS-Netz kann **offen, geschlossen** oder **gemischt** sein, je nachdem, ob die Anzahl der Aufträge im WS-Netz variabel für alle Auftragsarten, fest oder nur für bestimmte Auftragsarten veränderlich ist.

Geschlossene WS-Netze lassen sich mit Hilfe von endlichen Markovketten analysieren.

3.3 DIE MATRIX DER ÜBERGANGSRATEN

3.3.1 DEFINITION DER MATRIX Q DER ÜBERGANGSRATEN

Die Definition erfolgt anhand derjenigen in (MUE 80) und (STE 78).

Für die hier betrachteten endlichen, homogenen, zeitkontinuierlichen, irreduziblen Markovketten ist ein zeitunabhängiger Wert für den Zustandswechsel von i nach j angebbar, die **Rate q_{ij}**, mit der dieser Zustandswechsel erfolgt:

$$q_{ij} := \lim_{\Delta t \to 0} (p_{ij}(\Delta t))/\Delta t = \dot{p}_{ij}, \quad i \neq j.$$

Der gesamte "Abfluß" aus dem Zustand i ist also

$$- \sum_{j \neq i}^{n} q_{ij} ,$$

der "Zufluß" ist betragsmäßig genauso groß (weil eine endliche Markovkette, d. h. insbesondere ein geschlossenes System vorliegt), nur mit dem umgekehrten Vorzeichen.

Für einen Zustandsraum mit n Zuständen ist dann für genügend kleines Δt

$$1 - \sum_{j \neq i}^{n} q_{ij} \, P(\, X(t+\Delta t)=j \,)$$

die Wahrscheinlichkeit, daß im Zeitraum Δt der Zustand i beibehalten wird. Die Rate $- q_{ii}$, mit der der Zustand i verlassen wird, ist dann definiert als

$$- q_{ii} = \sum_{j \neq i}^{n} q_{ij} \quad .$$

Die sich ergebende Matrix $Q = (q_{ij})$ heißt Matrix der Übergangsraten. Sie ist genau dann irreduzibel, wenn die zugehörige Markovkette irreduzibel ist.

3.3.2 ZUSAMMENHANG VON Q MIT STOCHASTISCHEN MATRIZEN

Die Grundlage der Darstellung bildet (STE 78).

E bezeichne die Einheitsmatrix, $\underline{v}(t)$ den **Wahrscheinlichkeitsvektor** zum Zeitpunkt t mit

$$v_i(t) = P(X(t)=i),$$

$$\underline{v}(t) = (v_1(t), \ldots, v_n(t))^T .$$

Es gilt dann:

$$(Q^T \Delta t + E) \underline{v}(t) = \underline{v}(t + \Delta t),$$

was gleichbedeutend ist mit

$$\underline{v}(t)^T(Q \Delta t + E) = \underline{v}(t + \Delta t)^T,$$

komponentenweise:

$$v_i(t + \Delta t)$$

$$= \Delta t \sum_{j \neq i}^{n} v_j(t)\, q_{ji} \quad + \quad (\Delta t\, q_{ii} + 1)\, v_i(t)$$

$$= \Big(\sum_{j \neq i}^{n} q_{ji}\, v_j(t) \Big) \Delta t \quad + \quad v_i(t) \Big(1 - \sum_{j \neq i}^{n} q_{ij}\, \Delta t\Big) .$$

Falls eine **stationäre Grenzverteilung**

$$\underline{v} = \lim_{t \to \infty} \underline{v}(t)$$

existiert, so ergibt sie sich als Lösung des Eigenvektorproblems zum Eigenwert 1 der Matrix $Q\Delta t + E$, mit der Normierungsbedingung

$$\sum_{i=1}^{n} v_i = 1 \qquad .$$

Diese Matrix ist **zeilenstochastisch** (d. h. alle Zeilensummen sind gleich 1, und alle Komponenten sind nichtnegativ), falls

$$\Delta t \leq (\max_i |q_{ii}|)^{-1}.$$

Ist die Matrix Q irreduzibel, so auch $Q\Delta t + E$, und es gilt: 1 ist einziger Eigenwert vom Betrag 1 und außerdem auch einfacher Eigenwert (FHW 79), da mindestens ein Diagonalelement positiv ist. Im Sonderfall, daß alle Diagonalelemente von Q gleich sind, ist die Bedingung für Δt mit der strengen Ungleichung zu lesen.

Weil nur die Grenzverteilung gesucht ist, ist die Umwandlung der zeitkontinuierlichen Markovkette in eine zeitdiskrete mit den Zeiten 0, Δt, $2\Delta t$, ... erlaubt. Zur Bestimmung von $\underline{v}$ ist daher (mit $\Delta t = 1$) das homogene lineare Gleichungssystem

$$\underline{v}^T Q = 0 \quad <=> \quad Q^T \underline{v} = 0$$

mit der Bedingung

$$\sum_{i=1}^{n} v_i = 1$$

zu lösen.

Faßt man Q als Matrix der Zu- bzw. Abflußraten pro jeweiligem Zustand auf, dann stellt $\underline{v}$ den Vektor dar, der die Verteilung des Gesamtflusses im System im Gleichgewichtsfall angibt (d. h. für jeden Zustand heben sich Zu- und Abflüsse auf).

3.4 NUMERISCHE LÖSUNGSVERFAHREN

3.4.1 EINLEITUNG

In diesem Kapitel werden die verwendeten numerischen Verfahren beschrieben, wobei die genutzten Eigenschaften der Matrix der Übergangsraten genannt werden.

Um die übliche Darstellung der Verfahren zur Lösung von Gleichungssytemen zu erreichen, wird die transponierte Form Q_π^T von Q_π betrachtet[3] (s. auch Abschnitt 3.2).

Zuerst wird erläutert, wie die Singularität dieser Matrix beseitigt wird und welcher Zusammenhang der Lösung der transformierten Matrix mit dem gesuchten Eigenvektor besteht.

Zur iterativen Lösung des Problems wird eine Block-Gauß-Seidel Methode mit nachgeschalteter Aggregierung (**BGSA**) angewendet. Als Teil des Block-Gauß-Seidel (**BGS**) Verfahrens wird ein üblicher Gauß-Seidel (**GS**) Algorithmus verwendet. Für die Aggregierung nach (COU 77) wird noch eine direkte Methode zur Lösung von Gleichungssystemen mit speziellen Tridiagonalmatrizen benutzt.

In der Beschreibung werden einige Definitionen und Schreibweisen aus dem Kapitel über den Aufbau einer WS-Modellklasse schon verwendet, um den Zuschnitt der Verfahren auf das verwendete Problem zu unterstreichen.

3.4.2 TRANSFORMIERUNG DER MATRIX Q_π^T

Sei Q_π^T irreduzibel. Dann gilt nach Abschnitt 3.3, daß die Grenzverteilung $\underline{v}_\pi$ der Linkseigenvektor der Matrix $Q_\pi \Delta t + E$ zum einfachen Eigenwert 1 ist. Demnach hat Q_π^T den Rangabfall 1, ist also singulär.

Die ersten n-1 Zeilen von Q_π^T sind linear unabhängig. Q_π^T selbst ist (schwach) diagonaldominant. Ersetzt man q_{nn} durch $\tilde{q}_{nn} > |q_{nn}|$, dann ist die so entstandene Matrix $\tilde{Q}_\pi^T$ bzw. $\tilde{Q}_\pi$ irreduzibel diagonaldominant, da nun die strenge Diagonaldominanz

---

[3] Zur Bedeutung der Schreibweisen s. Definition: Partition π, Q_π, $\underline{y}_\pi$.

für ein Diagonalelement gilt. Demnach (BLU 72) ist $\tilde{Q}_\pi$ nicht singulär.

Setzt man nun $\underline{\tilde{b}} = (0, \ldots, \tilde{b}_n)^T$ mit $\tilde{b}_n \neq 0$, so ist

$$\tilde{Q}_\pi^T \, \underline{\tilde{v}} = \underline{\tilde{b}}$$

ein eindeutig lösbares Gleichungssystem. Zwischen diesem und dem Gleichungssystem

$$Q_\pi^T \, \underline{v} = 0$$

besteht nach (MUE 80) der Zusammenhang

$$\underline{v} = \underline{\tilde{v}} \; / \; || \, \underline{\tilde{v}} \, ||_1 \quad .$$

Die folgenden Lösungsverfahren arbeiten entsprechend alle mit dem transformierten Gleichungssystem.

3.4.3 DAS GAUß-SEIDEL VERFAHREN

Bei Anwendung der Block-Gauß-Seidel Methode wird es letztendlich erforderlich, punktweise zu iterieren. Es wurde dafür das Gauß-Seidel Verfahren gewählt.

Sei $A\underline{x}=\underline{b}$ das zu lösende Gleichungssystem, $A=(a_{ij})$ von der Ordnung (n,n). Sei D diejenige Matrix, die in der Diagonalen die Diagonalelemente von A enthält und deren restliche Elemente gleich 0 sind, L eine untere ("lower"), U eine obere ("upper") Dreiecksmatrix mit $A = L + D + U$. Sei kein Diagonalelement von A gleich 0, D und damit D+L also nicht singulär.

Eine Folge von iterierten Lösungsvektoren $\underline{x}^{(m)}$ erhält man über

$$(D + L) \, \underline{x}^{(m+1)} = U \, \underline{x}^{(m)} + \underline{b}$$

$$\Leftrightarrow$$

$$\underline{x}^{(m+1)} = (D + L)^{-1} \, U \, \underline{x}^{(m)} + (D + L)^{-1} \, \underline{b},$$

komponentenweise:

$$x_i^{(m+1)} = \Big(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)} \Big) \; / \; a_{ii} \, .$$

3.4.4 DAS BLOCK-GAUß-SEIDEL VERFAHREN

Anders als beim Gauß-Seidel Punktiterationsverfahren wird im Iterationsschritt m+1 nicht nur sukzessive je eine einzige Komponente x_i mit Hilfe aller anderen Werte bestimmt, sondern eine ganze Gruppe x_i bis x_j von Komponenten wird als unbekannt angenommen. Die übrigen werden wieder als bekannt vorausgesetzt. Man muß daher pro Iterationsschritt so viele Gleichungssysteme

lösen, wie Gruppen von Komponenten von $\underline{x}$ vorhanden sind, allerdings mit dem Vorteil, daß die Größe der Systeme geringer ist.

Sei $A_\pi \underline{x}_\pi = \underline{b}_\pi$ das zu lösende Gleichungssystem, A_π von der Ordnung (n,n). Sei D_π diejenige Matrix, die in der Diagonalen die sich anhand der Partition π ergebenden Blockdiagonaluntermatrizen von A_π enthält und deren sonstige Elemente Null sind, L_π und U_π jeweils untere und obere Blockdreiecksmatrizen mit $A_\pi = L_\pi + D_\pi + U_\pi$. Sei D_π und damit $D_\pi + L_\pi$ nicht singulär.

Eine Folge von iterierten Lösungsvektoren $\underline{x}_\pi^{(m)}$ erhält man über

$$(D_\pi + L_\pi)\, \underline{x}_\pi^{(m+1)} = U_\pi\, \underline{x}_\pi^{(m)} + \underline{b}_\pi$$

$$<=>$$

$$\underline{x}_\pi^{(m+1)} = (D_\pi + L_\pi)^{-1}\, U_\pi\, \underline{x}_\pi^{(m)} + (D_\pi + L_\pi)^{-1} \underline{b}_\pi$$

Betrachtet man speziell die nicht singuläre Matrix $\tilde{Q}_\pi^T$ und bezeichnet die Blockuntermatrizen dieser transponierten Matrix mit $\tilde{Q}_{IJ}$, dann vereinfacht sich das Verfahren wegen der vorhandenen Blocktridiagonalität: [4]

$$\tilde{Q}_{I,I}\, \underline{\tilde{v}}_I^{(m+1)} = -\tilde{Q}_{I,I-1}\, \underline{\tilde{v}}_{I-1}^{(m+1)} - \tilde{Q}_{I,I-1}\, \underline{\tilde{v}}_{I+1}^{(m)}$$

für $I \in {}_1\mathbb{N}_{N-1}$,

$$\tilde{Q}_{0,0}\, \underline{\tilde{v}}_0^{(m+1)} = -\tilde{Q}_{0,1}\, \underline{\tilde{v}}_1^{(m)} \quad \text{und}$$

$$\tilde{Q}_{N,N}\, \underline{\tilde{v}}_N^{(m+1)} = \underline{\tilde{b}}_N - \tilde{Q}_{N,N-1}\, \underline{\tilde{v}}_{N-1}^{(m+1)}.$$

Es ergeben sich N+1 Gleichungssysteme jeweils der Ordnung $(n(I),n(I))$.

3.4.5 AGGREGIERUNGSVERFAHREN FÜR NCD - MATRIZEN

Die wesentlichen Teile dieses Abschnitts gehen auf (COU 77) zurück.[5]

Eine Matrix A heißt **reduzibel (zerlegbar, decomposable)**, falls sie durch ein- und dieselbe Permutation der Zeilen- und Spaltenindizes in die Form

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

gebracht werden kann.

Ist A_{12} ebenfalls eine Nullmatrix, so heißt A **vollständig zerlegbar (completely decomposable)**.

---

[4] Nach (MUE 80) hat das BGS Verfahren daher außerdem eine doppelt so hohe Konvergenzrate wie das Block-Jacobi Verfahren.

[5] Ein Zahlenbeispiel des Verfahrens findet sich in (BOA 82).

Falls die stochastische Koeffizientenmatrix $A_\pi = (a_{ij})$ eines linearen Gleichungssystems zwar nicht reduzibel ist, sich jedoch als

$$A_\pi = A_\pi^* + \varepsilon C_\pi$$

schreiben läßt, wobei gilt:

- alle Matrizen sind anhand der Partition π strukturiert,
- A_π^* ist eine Blockdiagonalmatrix mit zeilenstochastischen, irreduziblen Matrizen in der Diagonalen (Bezeichnung: A_{II}^*),
- C_π ist eine Matrix, deren Zeilensummen Null ergeben und die außerhalb der Blockdiagonalen die Komponenten a_{ij}/ε enthält,
- ε ist eine positive, reelle Zahl, die sich als Maximum aller Zeilensummen von A_π ohne Berücksichtigung wiederum der Blockdiagonalen ergibt,

und: ε ist hinreichend klein - was noch näher präzisiert werden wird -,

dann heißt A_π **fast vollständig zerlegbar (nearly completely decomposable, NCD - Matrix).**

Anders formuliert (MUE 80): A_π heißt NCD - Matrix, falls es eine Matrix C_π^* gibt, so daß

$$A_\pi^* = A_\pi - C_\pi^*$$

vollständig zerlegbar ist und die Zeilensummennorm ε von C_π^* klein ist.

Der **maximale Kopplungsgrad** ε ermöglicht Aussagen über die Zulässigkeit und Genauigkeit der Aggregierung. Er ist ein Maß für den Maximalgrad an Wechselwirkungen zwischen den verschiedenen, durch die A_{II} dargestellten Untersystemen. Je näher ε bei Null liegt, desto geringer sind die Abweichungen zwischen den Eigenwerten von A_{II} und A_{II}^*, umso "ähnlicher" ist die Grenzverteilung von A_{II}^* als Linkseigenvektor zum größten Eigenwert dem entsprechenden Eigenvektor von A_{II}.

Sei $\lambda^*(2_I)$ der zweitgrößte Eigenwert der Matrix A_{II}^*. Eine hinreichende Bedingung für die fast-vollständige Zerlegbarkeit einer Matrix A ist nach (COU 77)

$$\varepsilon < (1 - \max_I(|\lambda^*(2_I)|)) / 2 .$$

Nach Simon und Ando (SIA 61), zitiert nach (COU 77), kann man für Systeme, die durch NCD-Matrizen beschreibbar sind, folgende vier Phasen unterscheiden:

- "short term dynamics": die Werte des Zustandswahrscheinlichkeitsvektors $\underline{v}_\pi(t)$ verändern sich, bis zu einem Zeitpunkt t_1, wo das
- "short term equilibrium" erreicht wird. Jetzt stehen die Komponenten des Vektors $\underline{v}_\pi(t)$ innerhalb jeder einzelnen Gruppe I schon etwa im gleichen Verhältnis wie diejenigen der stationären Grenzverteilung $\underline{v}_\pi$. Dabei stimmen die Relationen zwischen den Gruppen noch nicht mit der stationären Lösung überein. Es gilt daher:

$$\underline{v}_I(t)\, Y_I(t) = \lim_{t \to \infty} \underline{v}_I(t)$$

Nun ist die Phase der

- "long term dynamics" erreicht, $\underline{v}_\pi(t)$ strebt der stationären Verteilung $\underline{v}_\pi$, dem
- "long term equilibrium" zu, ohne daß sich noch größere Änderungen der Relationen der Werte innerhalb der einzelnen Gruppen ergäben.

Sei A_π von der Ordnung (n,n) und habe N+1 Blockdiagonaluntermatrizen, wie durch π beschrieben.

Liegen jetzt für jedes Untersystem A_{II} die Grenzverteilungen $\underline{v}_I$ des "short term equilibrium" vor, dann ergibt sich zur Bestimmung der N+1 Komponenten des Vektors $\underline{Y}$ ein Gleichungssystem von n Gleichungen in N+1 Unbekannten.

Addiert man jeweils alle Gleichungen mit gleichem Y_I, so erhält man ein Gleichungssystem mit einer Koeffizientenmatrix G der Ordnung (N+1,N+1): dies Vorgehen heißt **Aggregation**, das resultierende Gleichungssystem **aggregiertes Gleichungssystem**. Ein Koeffizient g_{IJ} von G ist dann:

$$g_{IJ} = \underline{v}_I \, A_{IJ} \, \underline{e}_{n(J)},$$

wobei $\underline{e}_{n(J)}$ ein Spaltenvektor mit n(J) Komponenten ist, die alle gleich 1 sind ($\underline{e}_{n(J)}$ drückt die Aufsummierung der Gleichungen aus).

Sei jetzt $A_\pi = (Q_\pi \Delta t + E)$, $A^*_{II} = Q^*_{II} + E$. Dann bringt nach (MUE 80) die Anwendung des Aggregierungsverfahrens selbst dann noch Vorteile, d. h. schnellere Konvergenz der gleichzeitig verwendeten Iteration, wenn die Anforderungen bezüglich ε und der Irreduzibilität der eben definierten Blockdiagonaluntermatrizen A^*_{II} nicht erfüllt sind.

Nach (MUE 80) stört auch die auftretende nichtstochastische Untermatrix A_{NN} nicht, denn:

die wesentliche, bei der Aggregierung genutzte Eigenschaft ist, daß ein Vektor $\underline{v}^*_\pi$ existiert, für den die Beziehung $\underline{v}^*_I Y_I = \underline{v}_I$ für jedes I gilt. In diesem Fall kann man auch auf die Stochastizität der Matrix verzichten, also $A = \tilde{Q}^T_\pi$ wählen.

Liegt, wie hier, ein inhomogenes Gleichungssystem mit der Inhomogenität $\underline{\tilde{b}}_\pi$ vor, wo für die Lösung $\underline{\tilde{v}}_\pi$ die Beziehung $\underline{v}^*_I Y_I = \underline{\tilde{v}}_I$ gilt, so erhält man dieselben Koeffizienten g_{IJ}, muß allerdings noch die n(I) Komponenten von $\underline{\tilde{b}}_I$ für jedes I aufsummieren, $\underline{B} := (B_I)$ mit $B_I = \underline{\tilde{b}}^T_I \, \underline{e}_{n(I)}$.

Folgende Vereinfachung des Aggregierungsverfahrens wird vorgenommen: weil es, wie geschildert, nur auf die Relationen der Komponenten der jeweiligen Eigenvektoren zueinander ankommt und Iterationsverfahren das zeitliche Verhalten der untersuchten Systeme nachbilden, können als Näherungen für die gesuchten Grenzverteilungen der Untersysteme direkt die im vorhergehenden BGS Iterationsschritt gewonnenen Approximationen $\underline{\tilde{v}}^{(m)}_I$ (als $\underline{v}^*_I$) verwendet werden.

Die Approximation $\underline{\tilde{a}}^{(m)}_\pi$ des Gleichgewichtsvektors des Gesamtsystems ist über

$$\underline{\tilde{a}}^{(m)}_I = \underline{\tilde{v}}^{(m)}_I \, Y_I$$

zu erhalten.

3.4.6 LÖSUNG EINES GLEICHUNGSSYSTEMS IN TRIDIAGONALFORM

Dieses Lösungsverfahren orientiert sich am Beweis zu Satz 5.2 in (FHW 79).

Bei Anwendung des geschilderten vereinfachten Aggregierungsverfahrens ist das Gleichungssystem $G^T\underline{Y} = \underline{B}$ zu lösen. Dabei hat G Tridiagonalgestalt, alle $g_{I,I+1}$ und alle $g_{I,I-1}$ sind positiv. Dies liegt an den Eigenschaften von $\tilde{Q}_\pi^T$ und einer geeigneten Wahl von $\tilde{\underline{v}}_\pi^{(0)}$.

Der einfacheren Schreibweise wegen sei nun $r_I := g_{I,I-1}$, $p_I := g_{I,I+1}$ und ein Diagonalelement von G^T sei d_I. Die Werte der oberen Nebendiagonalen sind dann wegen der Transponierung $r_0, r_1, \ldots, r_N$, die der unteren $p_1, p_2, \ldots, p_{N+1}$.

Weil alle Spaltensummen in $\tilde{Q}_\pi^T$, bis auf die von Spalte n, gleich Null sind, gilt:

$$d_0 = -p_1, \ \ldots, \ d_I = -(r_{I-1}+p_{I+1}), \ \ldots, \ d_{N-1} = -(r_{N-2}+p_N).$$

Wegen der Transformation von Q_π^T nach $\tilde{Q}_\pi^T$ und wegen der Konstruktion von G ist

$$d_N = -r_{N-1} + \tilde{v}_n(\tilde{q}_{nn}-q_{nn}).$$

Daraus folgt: $Y_I = (r_I/p_{I+1})\, Y_{I+1}$ für $I \in \mathbb{N}_{N-1}$, $Y_{N+1} = B_N/(r_{N-1}+d_N) = \tilde{b}_n/(\tilde{v}_n(\tilde{q}_{nn}-q_{nn}))$.

Anmerkung: Gilt $\tilde{q}_{nn}=\tilde{b}_n$ und sind beide sehr groß gewählt, z. B. als größte im Rechner darstellbare Zahl, so ist das im BGS Verfahren berechnete $\tilde{v}_n=1.0$ und q_{nn} vernachlässigbar, also $Y_{N+1}=1.0$.

Außerdem zeigt sich, daß die g_{II} bei der Aggregierung nicht explizit berechnet werden müssen.

4.0 AUFBAU EINER WS MODELLKLASSE

Im wesentlichen beruht dieses Kapitel auf entsprechenden Teilen der Dissertation von B. Müller (MUE 80).

4.1 EINLEITUNG

Die zur Beschreibung eines geschlossenen WS-Netzes notwendigen Begriffe werden eingeführt. Es wird unterschieden zwischen dem statischen Aufbau des Netzes und den Zuständen und Zustandsübergängen.

Davon ausgehend wird angegeben, wie sich zu jedem möglichen Zustand ein eindeutiger Index für die Matrix Q der Übergangsraten bestimmen läßt und wie sich deren Werte berechnen lassen.

Eine günstige Struktur der Matrix Q läßt sich über eine Zerlegung des WS-Netzes erreichen. Auf welche Weise dies geschieht, wird erläutert. Warum und wozu die erzielte Form der Matrix geeignet ist, ergibt sich aus dem Abschnitt 3.4 .

4.2 VERWENDETE BEGRIFFE

Definition: WS-Modell S

Ein WS-Modell S der Modellklasse WSM wird definiert durch:

- N : die Anzahl der Aufträge im System,
- M : die Anzahl der Stationen,
- W : eine spaltenstochastische Wechselmatrix der Ordnung (M,M). Ein Eintrag w_{ji} in dieser Matrix bedeutet, daß ein Übergang eines Auftrags von der Station i zur Station j mit der Wahrscheinlichkeit w_{ji} möglich ist. Alle Aufträge werden als nur zu einer Klasse gehörig betrachtet.
- T : eine Menge von Funktionen $T^{(i)}$, die alle als Typ der Station i bezeichnet werden. Jede Funktion hat als Wertebereich eine Menge von reellwertigen Matrizen und ist über der Anzahl n_i der Aufträge definiert. Jede Matrix $T^{(i)}(n_i)$ ist von der Ordnung (l_i+2, l_i+2), wobei die Indizes aus der Menge $\mathbb{N}_{l_i+1}$ sind. l_i+1 spezifiziert die Anzahl der möglichen internen Zustände einer Station i. Aus rechentechnischen Gründen wird ein weiterer, fiktiver Zustand eingeführt, daher ist die angegebene Ordnung der Matrizen größer als sich aus der Anzahl der internen Zustände ergeben würde.

- μ : eine Menge reellwertiger Funktionen $\mu^{(i)}$, die abhängig vom Zustand des WS-Netzes die Bedienraten der Station i festlegen.

  Sie sind interpretierbar als Parameter der Verteilung $1 - \exp(-\mu^{(i)}(z)\ t)$, wobei z ein - im Anschluß definierter - Zustand von S ist.

Definition: Zustand von S, Z(S).

Ein Zustand z des WS-Netzes S wird beschrieben durch ein 2*M - Tupel $(n_1, n_2, \ldots, n_M, c_1, c_2, \ldots, c_M)$. Dabei legen n_i und c_i den Zustand der Station i fest.

n_i gibt die Anzahl der Aufträge an der Station i an.

c_i spezifiziert den internen Zustand der Station i, $c_i \in \mathbb{N}_{1_i}$.

Weiterhin muß für alle Zustände z gelten:

$$\sum_{i=1}^{M} n_i = N$$

und

$n_i = 0 \iff c_i = 0$

(Definition des Ruhezustands einer Station.)

Die Menge aller Zustände von S wird mit Z(S) bezeichnet.

Definition: Zerlegung von S

Sei $S \in WSM$ und Stat(S) die Menge der M Stationen.

Eine Zerlegung von S in Teilsysteme S_1 und S_2 zur Zahl k wird definiert durch

$Stat(S_1) = \{$Station 1, ..., Station k-1$\}$

und

$Stat(S_2) = \{$Station k, ..., Station M$\}$.

Definition: Feinzustand, Jacksonzustand, Grobzustand.

Jedes $z \in Z(S)$ heißt Feinzustand von S.

Zerlegt man Z(S) so, daß alle z mit gleicher Auftragsverteilung in eine Menge fallen, dann heißt jede der entstehenden Mengen Jacksonzustand von S, Bezeichnung:

$$J(z) := \langle n_1, \ldots, n_M \rangle$$

$$:= \{z \mid z=(n_1', \ldots, n_M', c_1, \ldots, c_M),\ n_1'=n_1, \ldots, n_M'=n_M\}.$$

Ein Grobzustand G_I, $I \in \mathbb{N}_N$, bezüglich der Zerlegung von S zur Zahl k in Teilsysteme S_1 und S_2 ist die Menge aller Zustände $z \in Z(S)$, deren gesamte Auftragszahl im Teilsystem S_2 gerade I ist:

$$G_I := \left\{ z \in Z(S) \mid \sum_{i=k}^{M} n_i = I \right\}$$

$$= \left\{ z \in Z(S) \mid \sum_{i=1}^{k-1} n_i = N - I \right\}.$$

Definition: Partition π, Q_π, $\underline{y}_\pi$.

Die Anzahl der Feinzustände von G_I sei mit n(I) bezeichnet. Sei Z(S) in der Reihenfolge der Grobzustände $G_0, \ldots, G_N$ und darin jeweils lexikographisch absteigend geordnet. Mit $n := |Z(S)|$ sei $\mathbb{N}_n$ die zugehörige Indexmenge.

Eine Partition $\pi = (\pi_1, \ldots, \pi_N)$ dieser Indexmenge und damit von Z(S) ist gegeben durch

$$\pi_I := \sum_{J=0}^{I-1} n(J) + 1,$$

$$I \in \mathbb{N}_{N-1}.$$

π_I spezifiziert also die Position des ersten Feinzustands $z \in Z(S)$, der zum Grobzustand G_I gehört. Weil den Zeilen (und Spalten) der Matrix Q nach der eben angegebenen Ordnung Indizes zugeordnet werden, ist Q mit π in entsprechende Blockmatrizen zerlegt.

Wenn das WS-Netz von einem Zustand in einen anderen übergeht und dabei überhaupt eine Änderung der Auftragsverteilung stattfindet, so kann der Folgezustand nur dann in einem anderen Grobzustand liegen, wenn ein Auftrag von einem Teilsystem in das andere wechselte. Es sind demnach nur Wechsel in benachbarte Grobzustände möglich, Q_π hat daher Blocktridiagonalgestalt.

Q unter dieser Zerlegung wird mit Q_π bezeichnet.

Ein Vektor $\underline{y}$ mit $n = |Z(S)|$ Komponenten kann unter der Partition π in N+1 Teilvektoren $\underline{y}_I$ mit je n(I) Komponenten zerlegt werden. $\underline{y}_\pi$ ist dann die Verkettung dieser Teilvektoren, so daß $\underline{y}_\pi = \underline{y}$ ist.

Bevor weiter auf die einzelnen Bestandteile der Definition von S und auf die Konstruktion der Elemente von Q bzw. Q_π eingegangen werden kann, werden die möglichen Zustandsübergänge erläutert.

Definition: endogener und exogener Zustandsübergang.

Der Übergang von einem Zustand z in einen Zustand $z' \neq z$ ist auf zwei Arten möglich:

- endogen

  Ein interner Zustand c_i wird durch einen anderen, c_i', abgelöst:

  $(n_1, \ldots, n_M, c_1, \ldots, c_i, \ldots, c_M)$

  $\rightarrow (n_1, \ldots, n_M, c_1, \ldots, c_i', \ldots, c_M)$.

  Es findet sonst keine Veränderung, insbesondere des Jacksonzustands, statt.

- exogen

  Ein Auftrag wechselt von einer Station zu einer anderen:

  $(n_1, \ldots, n_i, \ldots, n_j, \ldots, n_M,$

  $c_1, \ldots, c_i, \ldots, c_j, \ldots, c_M)$

  $\rightarrow (n_1, \ldots, n_i-1, \ldots, n_j+1, \ldots, n_M,$

  $c_1, \ldots, c_i', \ldots, c_j', \ldots, c_M)$.

  Dies ist natürlich nur möglich, falls $n_i > 0$ ist. Folgende Fälle sind außerdem zu unterscheiden, getrennt nach Ausgangs- und Zielstation:
  - Fall I.1: $n_i = 1$. Es folgt: $c_i' = 0$, da $n_i - 1 = 0$.
  - Fall I.2: $n_i > 1$. $c_i' > 0$. Der genaue Wert bestimmt sich anhand des Typs der Station i.
  - Fall II.1: $n_j = 0$. Es folgt: $c_j' > 0$. Der genaue Wert bestimmt sich anhand des Typs der Station j.
  - Fall II.2: $n_j > 0$. $c_j' = c_j$, d. h. der interne Zustand wird durch die Ankunft eines weiteren Auftrags nicht verändert.

Im Rückgriff auf die Definition von $S \in WSM$ werden die Bestandteile davon näher betrachtet.

Definition: Wechselmatrix W.

Die Elemente w_{ji} der (M,M) Wechselmatrix W des WS-Netzes S sind Funktionen

$w_{ji} : Z(S) \rightarrow \mathbb{R}_0^+$, $i, j \in \mathbb{N}_M$

mit den Eigenschaften

- für alle $z \in Z(S)$ und für alle i gilt

$$\sum_{j=1}^{M} w_{ji}(z) = 1,$$

- für alle $z, z' \in Z(S)$ gilt

$$w_{ji}(z)>0 \iff w_{ji}(z')>0.$$

Die letzte Bedingung sichert, daß die Besetzungsstruktur der Wechselmatrix (und von Q_π) zustandsunabhängig bleibt.

Vereinfachend wird angenommen, daß alle w_{ji} konstant und damit unabhängig vom Systemzustand sind. (S. auch Beispiel 6.1.1)

Definition: Bedienrate μ.

Jeder Station $i \in \mathbb{N}_M$ ist eine Funktion

$$\mu^{(i)}: Z(S) \to \mathbb{R}_0^+$$

mit folgenden Eigenschaften zugeordnet:

$\mu^{(i)}(z) = 0$, falls $c_i = 0$,

$\mu^{(i)}(z) > 0$ sonst.

Für jede Station kann $\mu^{(i)}$ als Rate verstanden werden, mit der die internen Zustände dieser Station verändert werden, sofern kein exogener Zustandsübergang auftritt.

(S. auch Beispiel 6.1.1)

Definition: Typ $T^{(i)}$ einer Station i.

$T^{(i)}$ ist eine Funktion, die jedem $n_i \in \mathbb{N}_N$ eine $(0:l_i+1, 0:l_i+1)$ Matrix mit folgenden Eigenschaften zuordnet:

- $T^{(i)}(0)$ ist die Nullmatrix.

  Für alle $n_i>0$:

- $T^{(i)}(n_i) = (t^{(i)}_{jk}(n_i))$ ist eine nichtnegative, reelle Matrix,

- für alle $j \in \mathbb{N}_{l_i+1}$ gilt $t^{(i)}_{jj}(n_i) = 0$,

- $t^{(i)}_{0,l_i+1}(n_i) = 1$,

  $t^{(i)}_{l_i+1,0}(n_i) = 0$.

  Diese Bedingung ordnet dem - fiktiven - internen Zustand l_i+1 in Verbindung mit der späteren Angabe der Rate eines exogenen Zustandsübergangs die Bedeutung des Abgangs eines Auftrags von der Station i zu, und stellt sicher, daß ein im Ruhezustand der Station i ankommender Auftrag ($c_i=0$) die Station auch tatsächlich durchläuft.

- $T^{(i)}(n_i)$ ist spaltenstochastisch,

- $T^{(i)}(n_i)$ ist irreduzibel.

Weil nachfolgend für alle $n_i>0$ nur eine Matrix $T^{(i)}(n_i)$ definiert sein wird, wird sowohl sie als auch die Funktion $T^{(i)}$ als Typ der Station i bezeichnet werden.
(S. auch Beispiel 6.1.1)

4.3 DIE ZUORDNUNGSFUNKTION FZ

Um die Matrix Q_π der Übergangsraten zu erstellen, muß eine Zuordnung

fz:Z(S) -> **N**

der Zustände zu den Indizes der Zeilen (bzw. Spalten) der Matrix erfolgen.

Jedem 2*M - Tupel z **E** Z(S) muß gemäß der in der Definition der Partition π angegebenen Ordnung eineindeutig ein Index i zugeordnet werden. Als Hilfsmittel wird eine Baumstruktur verwendet.

Eine genaue Definition der graphentheoretischen Begriffe ist in (MUE 80), Kapitel 6.3, zu finden. An dieser Stelle erfolgt ein Überblick über das benutzte Verfahren.

Die benötigte Struktur setzt sich aus drei Teilbäumen zusammen, von denen zwei von derselben Art, nur mit unterschiedlichen Parametern sind.

Jeder Baum ist ein endlicher, bewerteter Vielwegbaum mit natürlichzahliger Bewertungsfunktion, d. h. es existieren nur endlich viele Knoten und Kanten, jede Kante trägt eine natürliche Zahl (das umfaßt die Null) als Bewertung und von jedem Knoten können beliebig viele Kanten ausgehen.

Werden die Knoten dieses Baumes bewertungsorientiert fallend durchlaufen, dann ergibt die dabei entstehende Reihenfolge der terminalen Knoten auch die Indexfolge (beginnend mit 1). "Bewertungsorientiert fallend" bedeutet hier, daß man von jeweils jedem besuchten Knoten diejenige wegführende Kante wählt, die noch bei keinem früheren Durchlauf ausgesucht worden ist und die höchste Bewertung von allen derartigen Kanten dieses Knotens hat.

* Begriffe: K, Stufe(K), AG(K), WE(K, $K^{(j)}$), WK(K).

Bezeichne K einen **Knoten** des J(N,M)-Baums, Stufe(K) die **Stufe** dieses Knotens. Die Stufe der Wurzel ist 1.

Bezeichne AG(K) die Anzahl der von K ausgehenden Kanten, den **Ausgangsgrad** von K. Sei WE(K, $K^{(j)}$) die **Bewertungsfunktion** der **Kante** zwischen den Knoten K und $K^{(j)}$ (falls dort eine existiert).

Sei WK(K) der **Wert** des Knotens K, d. h. WK(K) ist die Summe der Bewertungen aller Kanten auf dem Pfad von der Wurzel des Baumes bis zu K. Der Wert der Wurzel sei 0.

Sei N, M **E** **N**, M > 0.

* Definition: J(N,M)-Baum, G(N)-Baum.

In Bezug auf ein WS-Netz mit M Stationen und N Aufträgen läßt sich die Bewertung jeder Kante $(K, K^{(j)})$ im Baum als Anzahl der Aufträge n_i an der Station i = Stufe(K) interpretieren.

Dann bleiben am Knoten K mit Stufe(K)<M N-WK(K)+1 Möglichkeiten, die auf dieser Stufe noch übrigen Aufträge (das sind N-WK(K)) zu verteilen, an einem Knoten M - ter Stufe eine Möglichkeit. Die Knoten M+1 - ter Stufe haben dann alle den Wert N, die Aufträge sind alle verteilt, der Ausgangsgrad dieser Knoten ist daher 0.

Formal:

$$AG(K)=\begin{cases} N - WK(K) + 1, & \text{falls Stufe}(K) < M \\ 1, & \text{falls Stufe}(K) = M \\ 0, & \text{falls Stufe}(K) = M + 1 \end{cases}$$

$$WE(K, K^{(i)})= \begin{cases} AG(K) - i, & \text{falls Stufe}(K) < M \\ N - WK(K), & \text{falls Stufe}(K) = M \end{cases}$$

Ein Baum, der diese Bedingungen erfüllt, heißt J(N,M)-Baum.

Ein G(N)-Baum ist ein J(N,2)-Baum, entsprechend der Zerlegung des WS-Netzes in zwei Teilsysteme.

* Definition: FJ(z)-Baum.

In dem zu definierenden Baum hat jede Kante $(K, K^{(j)})$ die Bedeutung eines internen Zustands der Station i = Stufe(K).

Die Anzahl der Möglichkeiten für die internen Zustände der Station i hängt ab von n_i, der Zahl der Aufträge dort. Für n_i= 0 ist nur der Ruhezustand möglich, sonst alle übrigen. Entsprechend ist AG(K) für Stufe(K) M definiert:

$$AG(K)= \begin{cases} l_i, & \text{falls } n_i > 0 \\ 1, & \text{falls } n_i = 0 \end{cases}$$

Für Stufe(K) = M+1 ist AG(K)= 0.

Wegen der lexikographisch absteigenden Ordnung der Feinzustände muß die Bewertungsfunktion $WE(K, K^{(j)})$ ähnlich wie die des J(N,M)-Baums definiert sein:

$$WE(K, K^{(j)})= l_i - j + 1,$$
mit i = Stufe(K).

Insbesondere ist $WE(K, K^{(1)})= l_i$ für n_i=0. Weil aber keine andere Kante an diesem Knoten existiert, ist hier die Bewertung für das Durchlaufen des Baumes bedeutungslos.

Ein Baum, der die genannten Bedingungen erfüllt, heißt FJ(z)-Baum, da seine Gestalt durch den Jacksonzustand J(z) bestimmt wird.

4.4 DIE BERECHNUNG DER ÜBERGANGSRATEN

Es ist jetzt möglich, die Übergangsraten zwischen zwei Zuständen $z, z' \in Z(S)$ zu berechnen.

Sei fz die Funktion, die jedem $z \in Z(S)$ einen Indexwert für die Zeilen bzw. Spalten von Q_π zuordnet. Sei hier $Q_\pi^T = (q_{fz(z'),fz(z)})$.

* Die Rate $q_{fz(z'),fz(z)}$ eines endogenen Zustandsübergangs $z \to z'$ ist:

$$q_{fz(z'),fz(z)} = t^{(i)}_{c_i',c_i}(n_i)\ \mu^{(i)}(z).$$

* Die Rate eines exogenen Zustandsübergangs $z \to z'$ von Station i zu Station j setzt sich zusammen aus:

  - der Wahrscheinlichkeit w_{ji}, mit der ein Wechsel von Station i zu Station j möglich ist,
  - der Wahrscheinlichkeit, mit der jeweils die internen Übergänge an den Stationen erfolgen,
  - der Bedienrate $\mu^{(i)}$ der Station i, welche die Geschwindigkeit des Übergangs von einem internen Zustand der Station zu einem folgenden angibt.

Es wird dabei angenommen, daß die Station i direkt, d. h. ohne Verzögerung, in einen ihrer noch verbleibenden Auftragszahl entsprechenden internen Zustand übergeht.

Nach den schon unterschiedenen vier Fällen ergibt sich die Rate eines exogenen Zustandsübergangs wie folgt:

I.1 mit II.2, d. h. $n_i=1$, $c_i'=0$, $n_j>0$, $c_j=c_j'$:

$$q_{fz(z'),fz(z)} = w_{ji}(z) * t^{(i)}_{l_i+1,c_i}(n_i) * t^{(i)}_{0,l_i+1}(n_i) * \mu^{(i)}(z).$$

I.2 mit II.2, d. h. $n_i>1$, $c_i'>0$, $n_j>0$, $c_j=c_j'$:

$$q_{fz(z'),fz(z)} = w_{ji}(z) * t^{(i)}_{l_i+1,c_i}(n_i) * t^{(i)}_{0,l_i+1}(n_i) * t^{(i)}_{c_i',0}(n_i) * \mu^{(i)}(z).$$

Sollten die Fälle I.1 und I.2 nicht zusammen mit II.2, sondern mit II.1 ($n_j=0$, $c_j'>0$) auftreten, sind die obigen Produkte noch mit der Wahrscheinlichkeit des internen Übergangs der Station j,

$$t^{(j)}_{c_j',0}(n_j),$$

zu multiplizieren.

Aus dieser Berechnungsweise und der Irreduzibilität der Typmatrizen und der Wechselmatrix ergibt sich die Irreduzibilität von Q_π.

4.5 ZUSAMMENFASSUNG

Die für die Berechnung der Grenzverteilung aus der Matrix der Übergangsraten notwendigen Eigenschaften dieser Matrix ergeben sich aus der Definition des Modells. Die resultierenden Eigenschaften sind:

- Irreduzibilität aufgrund der Irreduzibilität der $T^{(i)}$ und W.
- Stochastizität von $Q\Delta t + E$ bzw. $Q_\pi \Delta t + E$ mit geeignetem Δt wegen dieser Eigenschaft von $T^{(i)}$ und W, den Eigenschaften der Bedienraten μ und der Definition der Diagonalelemente von Q bzw. Q_π.
- Blocktridiagonalität von Q_π nach Definition der Zuordnungsfunktion fz mit Hilfe von π.

5.0 DIE ALGORITHMEN

5.1 ÜBERBLICK

Aufbauend auf die bisher genannten Begriffe und Verfahrensbeschreibungen werden jetzt die konkreten Algorithmen soweit entwickelt, daß eine Umsetzung in eine beliebige andere Programmiersprache als die verwendete, FORTRAN IV, keinen großen Aufwand mehr erfordern sollte.

Einige durch FORTRAN IV erzwungene Einschränkungen gestalteten sich weniger gravierend unter Verwendung von COMMON-Blöcken. Dementsprechend sind für jeden Algorithmus die verwendeten COMMON-Bereiche aufgeführt. Außerdem wurde die Aufzählung der Parameter aufgespalten in solche expliziter und impliziter Art. Explizit ist ein im Aufruf der Funktion genannter Parameter, implizit einer, der für die Funktion entscheidend, nicht im Aufruf genannt und in einem COMMON-Block vorhanden ist.

Einleitend wird eine Überblicksdarstellung der wesentlichen Datenstrukturen und Variablen gegeben. Im Anschluß erfolgt die Beschreibung der Einzelroutinen.

Die Beschreibung der Algorithmen ist generell folgendermaßen aufgebaut:

- Name und Kurzbeschreibung der Funktion.
- Form des Aufrufs.
- Parameter: Bedeutung und Struktur, Zusammenhang mit schon bekannten Begriffen, insbesondere Umbenennungen.
- Genutzte COMMON-Bereiche.[6)]
- Externe Routinen: Name und Kurzbeschreibung.
- Funktionsablauf.
- Besonderheiten, z. B. Verhalten in Fehlerfällen.
- Struktogramme: Programmanweisungen werden umgangssprachlich oder in FORTRAN-ähnlicher Form geschrieben. Falls für eine Anweisung beides auftritt, geht der umgangssprachlichen Kommentierung ein "*" voraus.

---

6) Jeweils der erste Buchstabe des Namens einer COMMON Variable zeigt an, in welchem COMMON-Bereich die Variablen liegen, z. B. zeigt "B" die Zugehörigkeit von BN zum COMMON BFZS an.

5.2 DATENSTRUKTUREN

5.2.1 VORBEMERKUNGEN

Weil in FORTRAN IV keine Möglichkeit existiert, während der Programmausführung Speicherplatz anzufordern oder freizugeben, sind alle verwendeten Matrizen und Vektoren bezüglich bestimmter Maximalwerte der WS-Netz Parameter dimensioniert. Diese Maximalwerte selbst sind ebenfalls zugreifbar. Routinen, die bis zu diesen Grenzen eine dynamische Speicherplatzverwaltung nachbilden, vermeiden Indexbereichsüberschreitungen unter Verwendung dieser Werte.

Da in FORTRAN IV keine Möglichkeit besteht, Variablen verschiedenen Typs zu einer Struktur unter einem Namen zusammenzufassen, sind häufig zusammengehörige Daten in verschiedenen Variablen abgespeichert.

Zur Erleichterung der Programmentwicklung und der Lesbarkeit der Quelltexte wurden Konstanten, soweit sinnvoll, benannt. In FORTRAN IV können allerdings nur Variablen mit Namen versehen werden. Demzufolge wurden COMMON-Blöcke mit entsprechend initialisierten Variablen verwendet.

Alle COMMON-Blöcke wurden jeweils erst direkt vor der Kompilation als Kopie einer Prototyp-Definition in den Quelltext eingefügt, um abweichende Deklarationen und daraus resultierende Fehler auszuschließen.

Für FORTRAN IV sind nichtpositive Indizierungen unzulässig. Alle daher erforderlichen Indextransformationen wurden vorgenommen.

5.2.2 WICHTIGE STRUKTUREN UND VARIABLE

Die für das Verfahren wesentlichen Strukturen und Variablen sind, in Funktionsgruppen zusammengefaßt, aufgeführt.

* Alle WS-Netz Parameter.

  - BN, BM, BK, BLI, WST, WSW, WSMY entsprechen jeweils den Größen N, M, k, allen l_i, T, W, μ. Sie werden durch das Hauptprogramm eingelesen. Dimensionierungen orientieren sich an den Maximalwerten für N, M und l_i, jeweils BMAXN, BMAXM, BMAXLI.

* Z(S).

  - ANZ ist die Anzahl n= |Z(S)| der Feinzustände,

  - BZS(ANZ*BM*2) enthält alle 2*BM-Tupel z $\in$ Z(S), in der verlangten Ordnung.

  - AMAXZS ist die Maximalzahl an Elementen in BZS.

  - API ist ein Vektor mit BN+2 Elementen. Die Elemente API(2) bis API(BN+1) entsprechen den Komponenten π_1 bis π_N der Partition π. Zur Erleichterung der Programmierung wurden die Werte API(1)=1 und API(BN+2)=ANZ+1 hinzugenommen. Die Werte werden von FZ bestimmt.

* $\tilde{Q}_\pi^T$, v. Es werden nur die Nichtnullelemente der Matrix abgespeichert. Die Art der Abspeicherung erfolgt im Hinblick auf die benutzten Berechnungsverfahren. Alle Werte sind, soweit nicht anders angegeben, nach Aufruf von GNTRNS verfügbar.

- QRQ ist ein Vektor der Dimension AMAXPJ, der die Kehrwerte aller Diagonalelemente von $\tilde{Q}_\pi^T$ enthält. Diese Werte werden nur im GS Verfahren benötigt.

- QT ist ein Vektor der Dimension AMAXQT, der alle Nichtnullelemente der Blockdiagonaluntermatrizen von $\tilde{Q}_\pi^T$ enthält, bis auf die Diagonalelemente selbst. QT wird von GS benutzt. (S. Bild 5.2.2.)

- AQT ist genauso dimensioniert wie QT und enthält alle Spaltenindizes derjenigen Nichtnullelemente von $\tilde{Q}_\pi^T$, die in QT enthalten sind. (S. Bild 5.2.2.)

- QTE ist ein Vektor der Dimension AMAXQE und enthält alle Nichtnullelemente der Nebendiagonaluntermatrizen von $\tilde{Q}_\pi^T$. Zusammen mit denjenigen in QT und QRQ sind dies alle Nichtnullelemente der Matrix. QTE wird von BGS verwendet. (S. Bild 5.2.2.)

- AQTE ist so dimensioniert wie QTE und erfüllt für QTE die gleiche Funktion wie AQT für QT. (S. Bild 5.2.2.)

- AQTPJ ist (2, AMAXPJ) dimensioniert. AMAXPJ entspricht der maximal möglichen Zeilenzahl von $\tilde{Q}_\pi^T$, was gleich der maximal möglichen Anzahl von Zuständen ist. QT(AQTPJ(KPJBEG, J)) enthält das erste[7] Nichtnullelement der Zeile J in der Blockdiagonalen der Matrix $\tilde{Q}_\pi^T$, AQT an entsprechender Stelle den zugehörigen Spaltenindex. AQTPJ(KPJCUR, J) bezeichnet die Anzahl an Nichtnullelementen in der Zeile J der Blockdiagonalen von $\tilde{Q}_\pi^T$ (s. jedoch $LISTE). Wiederum bleiben Diagonalelemente unberücksichtigt. KPJBEG und KPJCUR sind Benennungen für Konstanten. (S. Bild 5.2.2.)

- AQTEPJ ist dimensioniert wie AQTPJ und erfüllt in gleicher Weise für QTE und AQTE dieselbe Funktion wie AQTPJ (s. auch $LISTE). (S. Bild 5.2.2.)

- QSKO ist ein Vektor der Länge AMAXPJ. Jedes Element enthält die Spaltensumme der zugehörigen Spalte der oberen Blocknebendiagonalen von $\tilde{Q}_\pi^T$. QSKO stellt demnach für $I \in \mathbb{N}_N$ die Verkettung aller Vektoren $(\tilde{Q}_{I,I-1}\ \underline{e}_{I-1})^T$ dar. QSKO wird von AGG genutzt.

- QSKU ist dimensioniert wie QSKO, erfüllt den entsprechenden Zweck für die untere Blocknebendiagonale und wird ebenfalls von AGG verwendet.

- QBS ist ein Skalar, enthält das einzige Nichtnullelement des Vektors $\tilde{b}$ und wird in BGS verwendet.

- QB ist ein Arbeitsbereich für BGS und GS von der Dimension AMAXQB.

- QG ist dimensioniert mit (2, BN+1) und ist Arbeitsbereich für AGG und TRI.

---

[7] Die Nichtnullelemente der betrachteten Zeile J sind in aufsteigender Reihenfolge der Spaltenindizes geordnet.

- QY besitzt BN+1 Elemente und ist Arbeitsbereich für AGG und TRI.

- QVN ist von der Dimension AMAXPJ und entspricht $\underline{y}_\pi$ bzw. $\tilde{\underline{y}}_\pi$. QVN wird verwendet von BGS, GS und AGG sowie vom Hauptprogramm zur Ausgabe des Ergebnisses.

* $LISTE, QLISTE. Aus den in Abschnitt 5.3.3.5 angegebenen Gründen ist das Anlegen einer Listenstruktur erforderlich. Die beiden Variablen $LISTE (INTEGER*4) und QLISTE (REAL) dienen dazu. $LISTE hat die Dimension (2, $LMAX), QLISTE die Dimension $LMAX.

Pro Zeile der Matrix $\tilde{Q}_\pi^T$ wird eine Liste der enthaltenen Nichtnullelemente angelegt. Die Anfänge der Listen stehen während der in Abschnitt 5.3.3.5 angegebenen Zeit in AQTPJ(KPJBEG, j) bzw. AQTEPJ(KPJBEG, j), die Endpunkte in AQTPJ(KPJCUR, j) bzw. AQTEPJ(KPJBEG, j). Dabei benennen KPJBEG und KPJCUR Konstanten, j bezeichnet einen Zeilenindex der Matrix $\tilde{Q}_\pi^T$. Die in AQTPJ festgehaltenen Listen enthalten jeweils alle Nichtnullelemente der Zeilen in den Blockdiagonaluntermatrizen, die in AQTEPJ jene der Nebendiagonaluntermatrizen.

Ein Element der Liste, z. B. das Anfangselement mit Index J:= AQTPJ(KPJBEG, j), wird dargestellt durch drei Werte:

- $LISTE(KLSTI, J) ist der zu Zeile j gehörige Spaltenindex i für dieses Listenelement.

- $LISTE(KLSTNX, J) bezeichnet den Index in $LISTE des nachfolgenden Listenelements.

- QLISTE(J) enthält das Element q_{ji} der Matrix $\tilde{Q}_\pi^T$.

Zur Anzeige des Endes der Liste dient die Konstante KNULL.

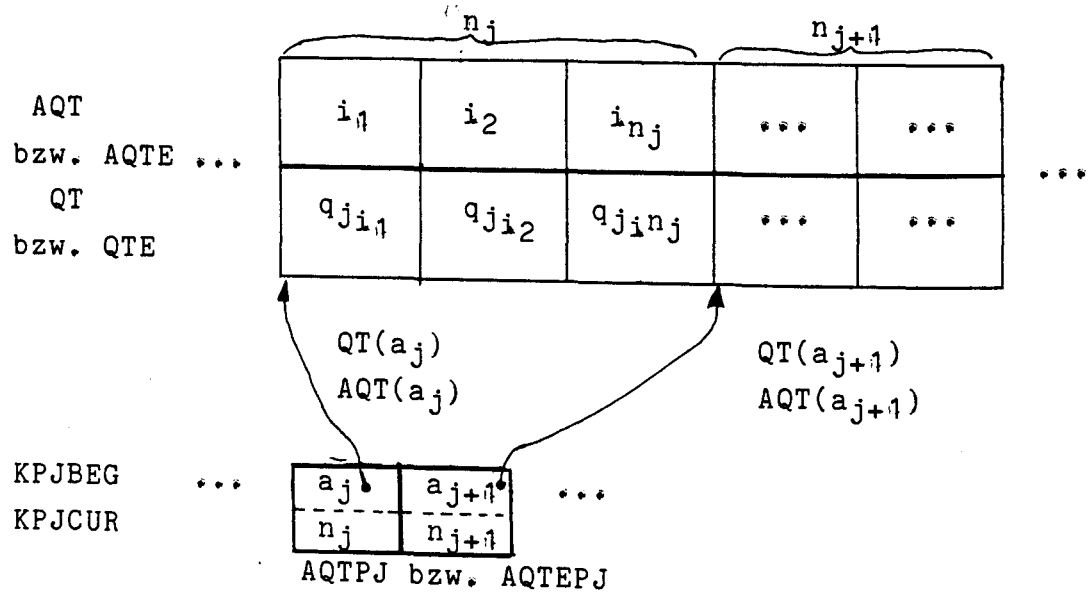


Bild 5.2.2: Struktur zur Abspeicherung der Matrix $\tilde{Q}_\pi^T$.

5.3 EINZELBESCHREIBUNGEN

5.3.1 HAUPTPROGRAMM WSNET UND HILFSROUTINEN

5.3.1.1 WSNET

* WSNET - Numerische Berechnung der Grenzverteilung eines Warteschlangennetzwerks.

* Aufruf: Systemabhängig.

* Parameter

- Eingabe:

BN	INTEGER*2, COMMON BFZS. Anzahl N der Aufträge im WS-Netz.
BM	INTEGER*2, COMMON BFZS. Zahl M der Stationen im WS-Netz.
BK	INTEGER*2, COMMON BFZS. Nummer k der Station, an der das WS-Netz zerlegt werden soll.
BLI	INTEGER*2, COMMON BFZS. Vektor der Maximallänge BMAXM, der aktuellen Länge BM. Enthält pro Station i die Anzahl l_i der internen aktiven Zustände.
WST	REAL, COMMON WSTMY. Feld der maximalen Dimension (BMAXM, BMAXLI+2, BMAXLI+2), davon aktuell genutzt pro Station i: (i, BLI(i)+2, BLI(i)+2). Entspricht der Menge T der Typen $T^{(i)}$ des WS-Netzes.
MYTP	INTEGER*2, COMMON MYCOM. Vektor der Maximallänge BMAXM und der aktuellen Länge BM. Enthält pro Station i einen Code für die Art der Abhängigkeit der Bedienrate $\mu^{(i)}$ vom Zustand des WS-Netzes. Der Code muß mit einem der gültigen Werte KCI, KFNI, KDNI oder KON übereinstimmen.
WSMY	REAL, COMMON WSTMY. Matrix der maximalen Dimension (BMAXM, BMAXLI). Aktuell genutzt wird pro Station i: (i, BLI(i)), falls MYTP(i)=KCI, sonst (i, 1). Entspricht den Bedienraten $\mu^{(i)}(z)$.
WSW	REAL, COMMON WSTMY. Matrix der maximalen Dimension (BMAXM, BMAXM), aktuell genutzt wird (BM, BM). Entspricht der Wechselmatrix W.

Alle Parameter werden eingelesen.

- Ausgabe:

BZS	INTEGER*2, COMMON BFZS. Vektor der aktuellen Länge ANZ*BM*2. Entspricht Z(S) mit der bei der Definition der Partition π angegebenen Ordnung.
QVN	REAL, COMMON Q. Vektor der aktuellen Länge ANZ. Enthält die berechnete Grenzverteilung.
ANJZ	INTEGER, COMMON AFZS, wird zu 0 initialisiert.
AQTCUR	REAL, COMMON AFZS, wird zu 1 initialisiert.
AQECUR	REAL, COMMON AFZS, wird zu 1 initialisiert.
$CURZ	INTEGER, COMMON $WRKSP, wird zu 1 initialisiert.
$CURSK	INTEGER, COMMON $WRKSP, wird zu 0 initialisiert.

* Genutzte COMMON-Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
MYCOM
Q
WSTMY
$WRKSP

* Externe Routinen

- FRIST: Gebe die restliche zur Verfügung stehende Rechenzeit an (Bibliotheksprogramm).
- ZEIT: Gebe die Tageszeit an (Bibliotheksprogramm).
- MSG: Gebe eine Fehlernachricht aus.
- FZ: Berechne fz.
- GNTRNS: Berechne alle Zustandsübergänge und ihre Raten.
- BGSA: Berechne die Grenzverteilung mit dem BGSA Verfahren.

* Funktionsablauf

Einige COMMON Variable werden initialisiert. Die WS-Netz Parameter werden eingelesen und die genannten Bedingungen werden abgeprüft, mit Ausnahme der Irreduzibilität aller $T^{(i)}$ und W. Die Maximalzahl der auszuführenden Iterationen wird eingelesen. Bei Auftreten von Eingabefehlern wird eine entsprechende Nachricht ausgegeben. Hilfsgrößen werden initialisiert.

Am Ende aller Eingaben wird eine vollständige Liste aller WS-Netz Parameter ausgegeben.

Danach wird fz berechnet (CALL FZ), die Liste der Zustände ausgegeben und die Matrix Q_π^T erstellt (CALL GNTRNS). Eine Belegungsstatistik für wichtige Variable wird erzeugt.

Anschließend wird die Transformation von Q_π^T zu $\tilde{Q}_\pi^T$ durchgeführt. $\underline{\tilde{v}}_\pi^{(0)}$ wird initialisiert.

Das BGSA Verfahren wird durchgeführt (CALL BGSA). Die Rücktransformation des Ergebnisses wird vorgenommen und das Resultat ausgegeben.

Der Zeitbedarf der Routinen FZ, GNTRNS und BGSA sowie des Hauptprogramms selbst (näherungsweise) wird aufgelistet.

* Besonderheiten

Fehlerhafte Eingaben können wiederholt werden, falls vor der Kompilation die Variable DIALOG mit .TRUE. initialisiert wird. Sonst führen sie zum Abbruch des Programms (s. STOP Codes).

Der Umfang der Ausgaben kann gesteuert werden, je nachdem wie die Variablen NOQ, NOQEL und NOZS gesetzt sind.

* Struktogramm

Initialisierung: Zeitnahme, Initialisierung einiger COMMON Variablen
Eingabe der WS-Netz Parameter
Eingabe der Maximalzahl an Iterationen
Ausgabe aller eingelesenen Werte
CALL FZ
Ausgabe der Partition und, falls gewünscht, der Zustände
Berechnung der Matrix Q_{π}^{T}: CALL GNTRNS
Erzeugen der Belegungsstatistik
Falls gewünscht und bei kleiner Zustandszahl, Ausgabe eines Bildes der Besetzungsstruktur der Matrix Q_{π}^{T}
Falls gewünscht, Ausgabe aller positiven Matrixelemente
Durchführen der Transformation
Startiterationsvektor bestimmen (QVN)
CALL BGSA
Rücktransformation durchführen
Ergebnis ausgeben
Zeitverbrauchsstatistik ausgeben

5.3.1.2 MSG

* MSG - Gebe eine Fehlernachricht aus.

* Aufruf: CALL MSG(MSGNO, DIALOG, &SNO)

* Parameter (nur explizite Parameter vorhanden)

- Eingabe

  MSGNO INTEGER, Nummer der auszugebenden Fehlernachricht.
  DIALOG LOGICAL*1, gibt die Aktion nach Ende der Ausgabe der Nachricht an: falls DIALOG=.TRUE. ist, wird zur angegebenen Statementnummer verzweigt, sonst STOP 0.
  &SNO muß eine den FORTRAN Regeln für Statementnummern entsprechende Zahl sein, der ein "&" vorausgeht. Bedeutung s. DIALOG.

* Genutzte COMMON-Bereiche

  Keine.

* Externe Routinen

  Keine

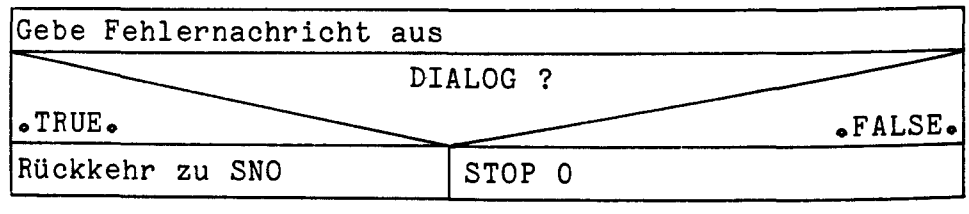
* Funktionsablauf

  Die verlangte Fehlernachricht wird ausgegeben und die dem Wert von DIALOG entsprechende Aktion ergriffen.

* Besonderheiten

  Falls zur angegebenen MSGNO keine Nachricht definiert ist, STOP 12.

* Struktogramm



5.3.1.3 FRIST

* FRIST - Bibliotheksroutine zur Feststellung der verbleibenden Rechenzeit.

* Aufruf: CALL FRIST(CPURST)

* Parameter

  CPURST REAL, Rückgabewert. Enthält die verbleibende Rechenzeit in Hundertstelsekunden.

  Weitere Einzelheiten unbekannt.

5.3.1.4 ZEIT

* ZEIT - Bibliotheksroutine zur Feststellung der Tageszeit.

* Aufruf: CALL ZEIT(REALZT)

* Parameter

REALZT REAL, gibt die Tageszeit bezüglich eines rechensystemabhängigen Basiswerts in Sekunden an. Zur Bestimmung von Realzeitdifferenzen geeignet.

Keine weiteren Einzelheiten bekannt.

5.3.2 FZ UND ABHÄNGIGE ROUTINEN

5.3.2.1 FZ

* FZ - Berechne fz.

* Aufruf: CALL FZ

* Parameter

- Explizite Parameter

Keine.

- Implizite Parameter

  - Eingabe:

BN INTEGER*2, Anzahl N der Aufträge im WS-Netz,
BM INTEGER*2, Zahl M der Stationen im WS-Netz,
BK INTEGER*2, Nummer k der Station, an der das WS-Netz zerlegt wird,

alle aus COMMON BFZS.

- Implizite Parameter

  - Ausgabe:

API INTEGER, Dimension (BN+2). Entspricht der Partition π mit den in Abschnitt 5.2.2 genannten Modifikationen.
ANI INTEGER, Dimension (BN+1). ANI(I) entspricht n(I-1).

Die Variablen gehören zum COMMON AFZS. Von FZ gerufene Unterprogramme ändern weitere COMMON Variable, insbesondere z. B. BZS.

* Genutzte COMMON-Bereiche

AFZS
BFZS
CKONST (Codierhilfe, bleibt unverändert).

* Externe Routinen

- FJZ: Durchlaufe FJ(z)-Baum.
- JNM: Durchlaufe J(N, M)-Baum.
- SETJZ: Lege Anfangsindex eines Jacksonzustands im Vektor AJZ ab.
- SETPI: Lege ein Element der Partition π, unter Berücksichtigung der genannten Modifikation (s. Abschnitt 5.2.2), im Vektor API ab.
- SETZS: Lege einen Feinzustand im Vektor BZS ab.

* Funktionsablauf

Für den Grobzustand 0 und den ersten Jacksonzustand werden die zugehörigen Indexwerte (Variable INDEX) eingetragen (CALL SETPI, CALL SETJZ).

Die Zerlegung des WS-Netzes ist mit Hilfe eines G(BN)-Baumes vorzunehmen. Dies wird durch einen Aufruf von JNM realisiert. Dabei wird angezeigt, daß eine Wurzel anzulegen ist (GINI=1) und die Auftragsverteilung für die Teilsysteme nicht abgespeichert werden soll (CNSET).

Bei Auffinden eines terminalen Knotens oder nach vollständiger Abarbeitung des Baumes kehrt JNM zu FZ zurück.

Ist der G(BN)-Baum komplett durchlaufen (GFRT≠0), dann sind alle Feinzustände gefunden und je ein Index zugeordnet, FZ ist fertig.

Sonst (GFRT=0) sind für den Grobzustand, der durch den gefundenen terminalen Knoten repräsentiert wird, alle möglichen Auftragsverteilungen zu konstruieren. Zuerst wird die Auftragsverteilung im Teilsystem S_1 durch einen Aufruf von JNM festgestellt (CSET). Dabei ist eine neue Wurzel anzulegen (JS1INI=1). Die Zahl der zu verteilenden Aufträge ist BN-I, die Anzahl der Stationen in S_1 ist BK-1.

Bei Auffinden eines terminalen Knotens oder nach vollständiger Abarbeitung des Baumes kehrt JNM zu FZ zurück.

Die Bedeutung der vollständigen Abarbeitung wird später beschrieben. Im übrigen Fall steht noch die Konstruktion aller Auftragsverteilungen im Teilsystem S_2 aus. Dies erfolgt durch einen erneuten Aufruf von JNM mit entsprechenden Parameterwerten.

Falls JNM einen terminalen Knoten gefunden hat, so ist nun der zur vollständigen Auftragsverteilung gehörige FJ(z)-Baum zu durchlaufen. Dies erledigt FJZ. Dabei ist keine besondere Anzeige zum Anlegen der Wurzel nötig, weil FJ(z)-Bäume nicht aneinandergefügt werden und daher jeder FJ(z)-Baum erst vollständig abzuarbeiten ist, bevor der nächste angelegt werden kann.

Bei Rückkehr von FJZ ist im Falle des Auffindens eines terminalen Knotens INDEX zu erhöhen und der gefundene Zustand an seiner Stelle in BZS einzutragen (CALL SETZS). Erneut ist FJZ aufzurufen.

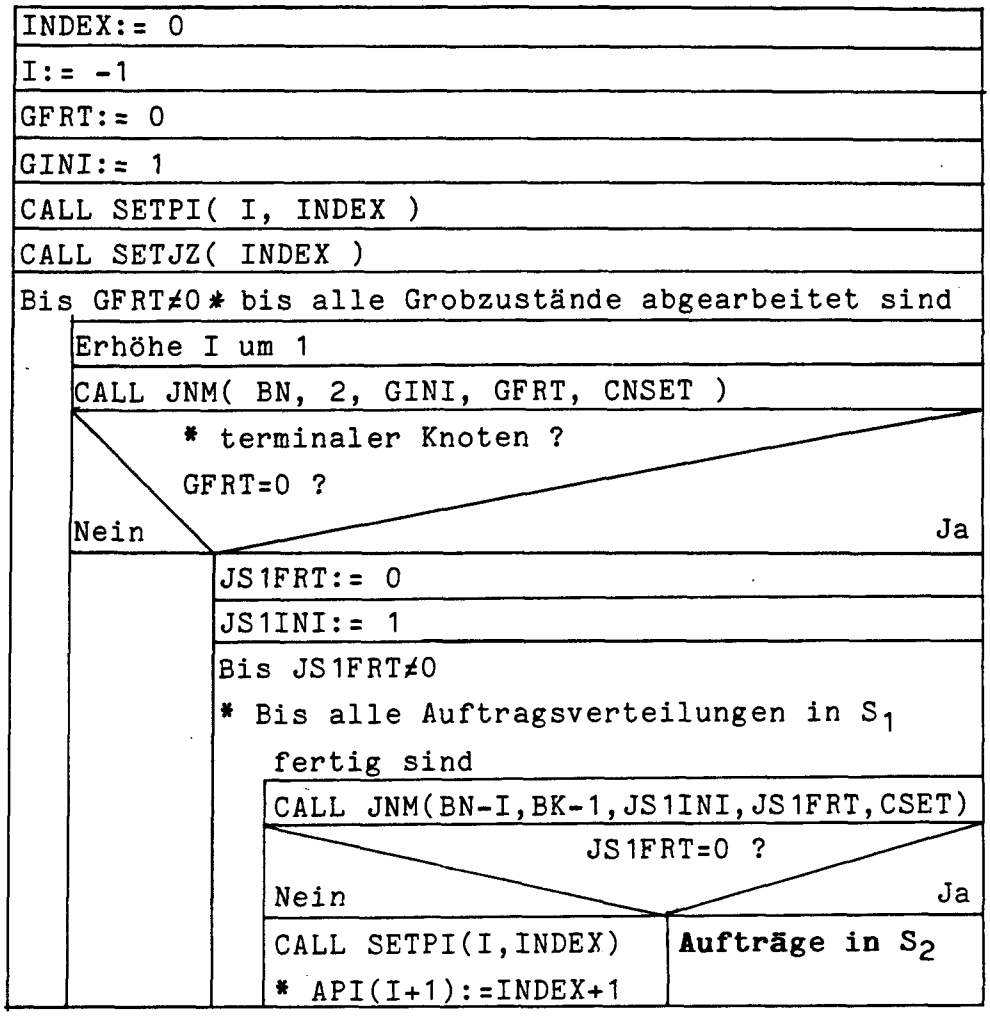
Sonst, nach Abarbeitung des gesamten FJ(z)-Baumes, ist der Anfang eines neuen Jacksonzustands als Hilfe für spätere Routinen zu markieren. Die Auftragsverteilung in S_2 ist neu zu treffen (CALL JNM).

Falls schon alle Verteilungen gefunden wurden, ist diejenige in S_1 zu ändern. Wenn auch dort der zugehörige J(BN-I,BK-1)-Baum abgearbeitet ist, enthält INDEX einen Wert der Partition π (CALL SETPI).

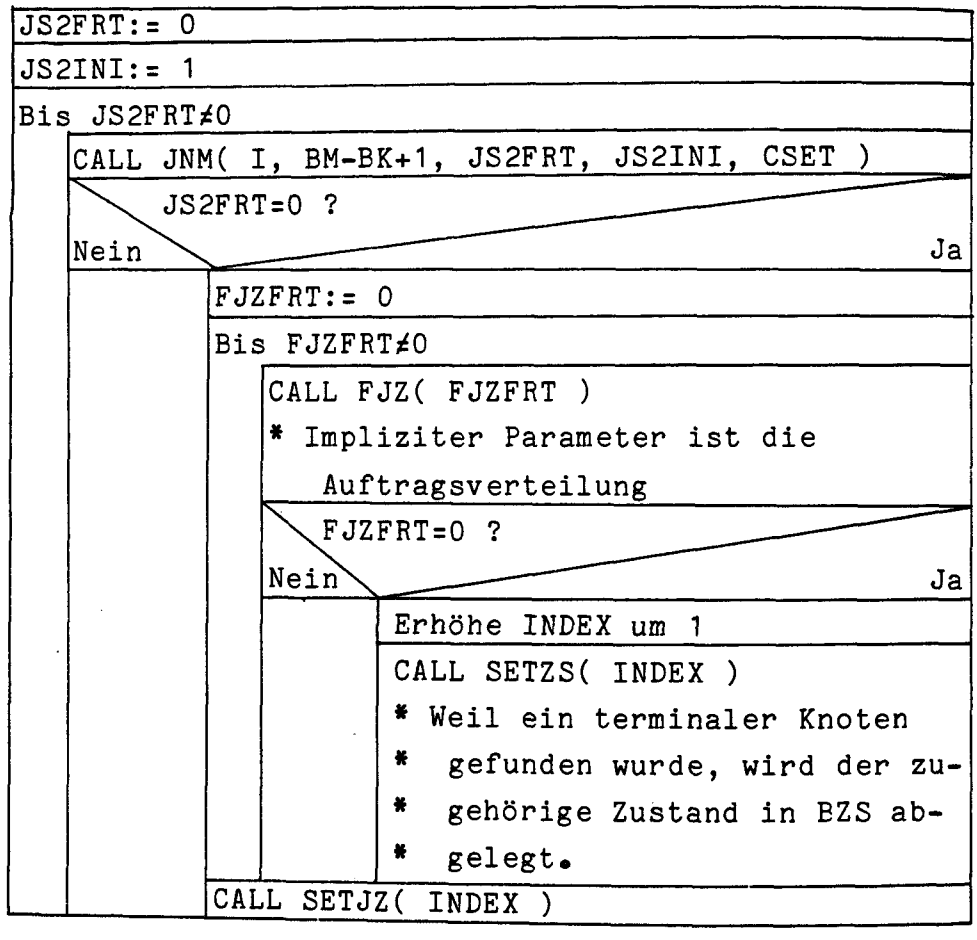
* Besonderheiten

Keine.

* Struktogramm



Aufträge in S_2



5.3.2.2 SETJZ

* SETJZ - Trage den Index des ersten Feinzustands eines Jacksonzustands in AJZ ein.

* Aufruf: CALL SETJZ(INDEX)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  INDEX INTEGER. Index des letzten zum vorhergehenden Jacksonzustand gehörenden Feinzustands.

- Implizite Parameter

  - Eingabe

    ANJZ INTEGER, aus COMMON AFZS. Index des letzten Jacksonzustands.

- Ausgabe

  ANJZ Nach Aufruf: Index des jetzigen Jacksonzustands.
  AMAXJZ INTEGER, Maximaldimension von AJZ, aus COMMON AFZS.
  AJZ INTEGER, aus COMMON AFZS, maximale Dimension AMAXJZ, aktuelle Dimension ANJZ. Nach Aufruf enthält AJZ(ANJZ) den Index des ersten Feinzustands des Jacksonzustands ANJZ.

* Genutzte COMMON Bereiche

  AFZS

* Externe Routinen

  Keine.

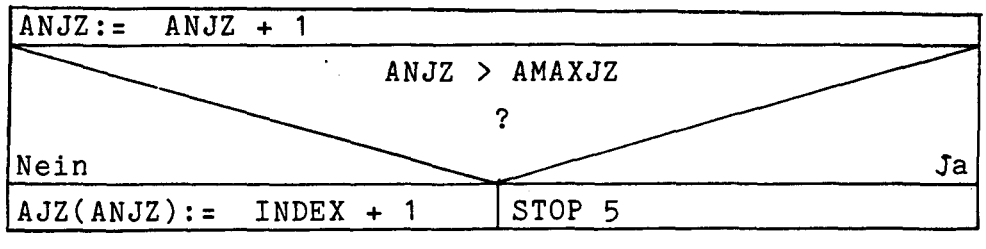
* Funktionsablauf

  ANJZ wird um 1 erhöht und danach AJZ(ANJZ) auf den Index des ersten Feinzustands zum Jacksonzustand mit der Nummer ANJZ gesetzt.

* Besonderheiten

  Falls die Maximaldimension von AJZ überschritten wird, STOP 5.

* Struktogramm



5.3.2.3 SETPI

* SETPI - Trage einen Index der Partition π in API ein.

* Aufruf: CALL SETPI(I,INDEX)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  I INTEGER, Index I des gerade verlassenen Grobzustands G_I.
  INDEX INTEGER, Index des letzten Feinzustands des Grobzustands G_I.

- Implizite Parameter (nur Ausgabe vorhanden)

  BMAXPI INTEGER*2, aus COMMON BFZS, Maximaldimension von API.
  API INTEGER, aus COMMON AFZS, maximale Dimension BMAXPI, aktuelle Dimension BN+2. Enthält nach Aufruf von SETPI an der Stelle API(I + 2) den Wert INDEX+1.

* Genutzte COMMON Bereiche

AFZS
BFZS

* Externe Routinen

Keine.

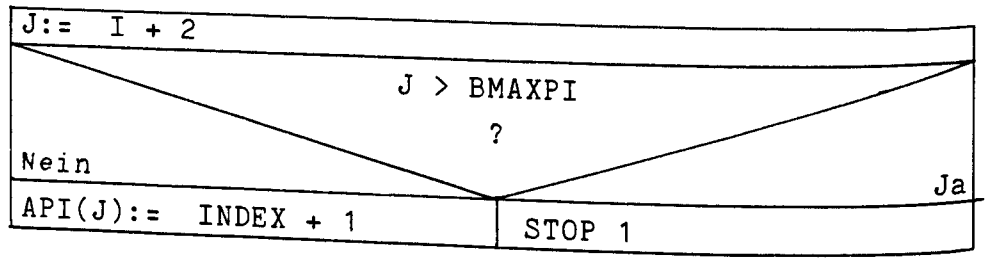
* Funktionsablauf

Weil abweichend von der Definition von auch zum Grobzustand G_0 der erste Index abgespeichert wird und in FORTRAN IV keine nichtpositiven Indizes erlaubt sind, ist I um 2 zu erhöhen und der neue Anfangsindex dieses Grobzustands, INDEX+1, auf API(I+2) zuzuweisen.

* Besonderheiten

Falls zu viele Grobzustände, STOP 1.

* Struktogramm



5.3.2.4 SETZS

* SETZS - Trage einen Zustand an seiner Stelle in BZS ein.

* Aufruf: CALL SETZS(INDEX)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  INDEX INTEGER. Index, laut fz, des einzutragenden Zustands, muß der in der Folge der bisherigen Aufrufe größte Wert sein.

- Implizite Parameter

  - Eingabe

    BM2 INTEGER*2, aus COMMON BFZS, ist 2*BM und spezifiziert die Länge des Tupels, das den Zustand des WS-Netzes darstellt.
    AMAXZS INTEGER, aus COMMON AFZS, höchster Startindex für ein Tupel von BZS.
    $Z INTEGER*2, Maximaldimension $MAXZ, aktuelle Dimension BM2, COMMON $WRKSP. Stellt den in BZS einzutragenden Zustand dar.

  - Ausgabe

    BZS INTEGER*2, aus COMMON BFZS, Maximaldimension AMAXZS + 2*BMAXM - 1, aktuelle Dimension ANZ*BM2. Stellt den Zustandsraum Z(S) dar. Ein z E Z(S) mit Index i wird repräsentiert durch die Elemente BZS((i-1)*BM2 + 1) bis BZS(i*BM2 - 1).
    ANZ INTEGER, COMMON AFZS. Entspricht n= |Z(S)|.

* Genutzte COMMON Bereiche

AFZS
BFZS
$WRKSP

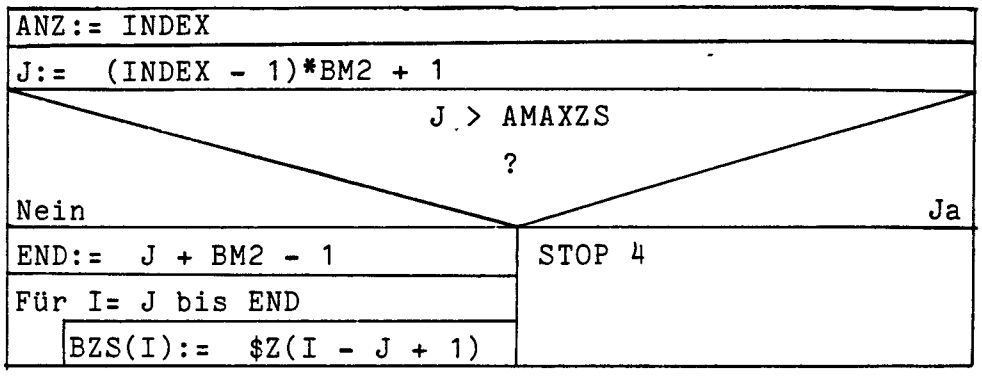
* Funktionsablauf

ANZ wird auf den neuen Höchstwert, INDEX, gesetzt. Die Position der ersten Komponente des neuen Zustands in BZS wird berechnet. Alle Komponenten des Tupels werden von $Z nach BZS übertragen.

* Besonderheiten

Falls die Maximaldimension von BZS im Laufe der Übertragung des Zustands von $Z nach BZS überschritten werden würde, STOP 4.

* Struktogramm



5.3.2.5 FJZ

* FJZ - Durchlaufe einen FJ(z)-Baum.

* Aufruf: CALL FJZ(FRT)

* Parameter

- Explizite Parameter (nur Ausgabe vorhanden)

  FRT INTEGER. Falls 0: terminaler Knoten gefunden, sonst Baum abgearbeitet.

- Implizite Parameter

  Keine.

* Genutzte COMMON Bereiche

CKONST (Codierhilfe, bleibt unverändert).

* Externe Routinen

- BACKZ: Entferne den letzten Eintrag aus dem Stack $Z.
- DKFJZ: Definiere einen Knoten eines FJ(z)-Baums.
- POPK: Hole einen Knoten aus dem Stack $STCKK.
- PUSHK: Bringe einen Knoten auf den Stack $STCKK.
- PUSHZ: Bringe einen Wert auf den Stack $Z.

* Funktionsablauf

Falls die Wurzel des FJ(z)-Baums anzulegen ist, werden die Parameter für DKFJZ entsprechend gesetzt und DKFJZ aufgerufen, die Variable KNOTEN ist dann initialisiert.

Sonst enthält KNOTEN noch die Werte des letzten gefundenen terminalen Knotens.

In beiden Fällen wird nun solange ein Knoten vom Stack geholt (CALL POPK) und die zugehörige Kantenbewertung eliminiert (CALL BACKZ), d. h. also im Baum "aufwärts", in Richtung Wurzel gelaufen, bis ein noch nicht vollständig abgearbeiteter Knoten gefunden wird oder der Stack leer ist (letzteres ist gleichbedeutend mit einem abgearbeiteten Knoten der Stufe 1).

Bei leerem Stack wird FRT auf 1 gesetzt, was bedeutet, daß der FJ(z)-Baum vollständig durchlaufen worden ist.

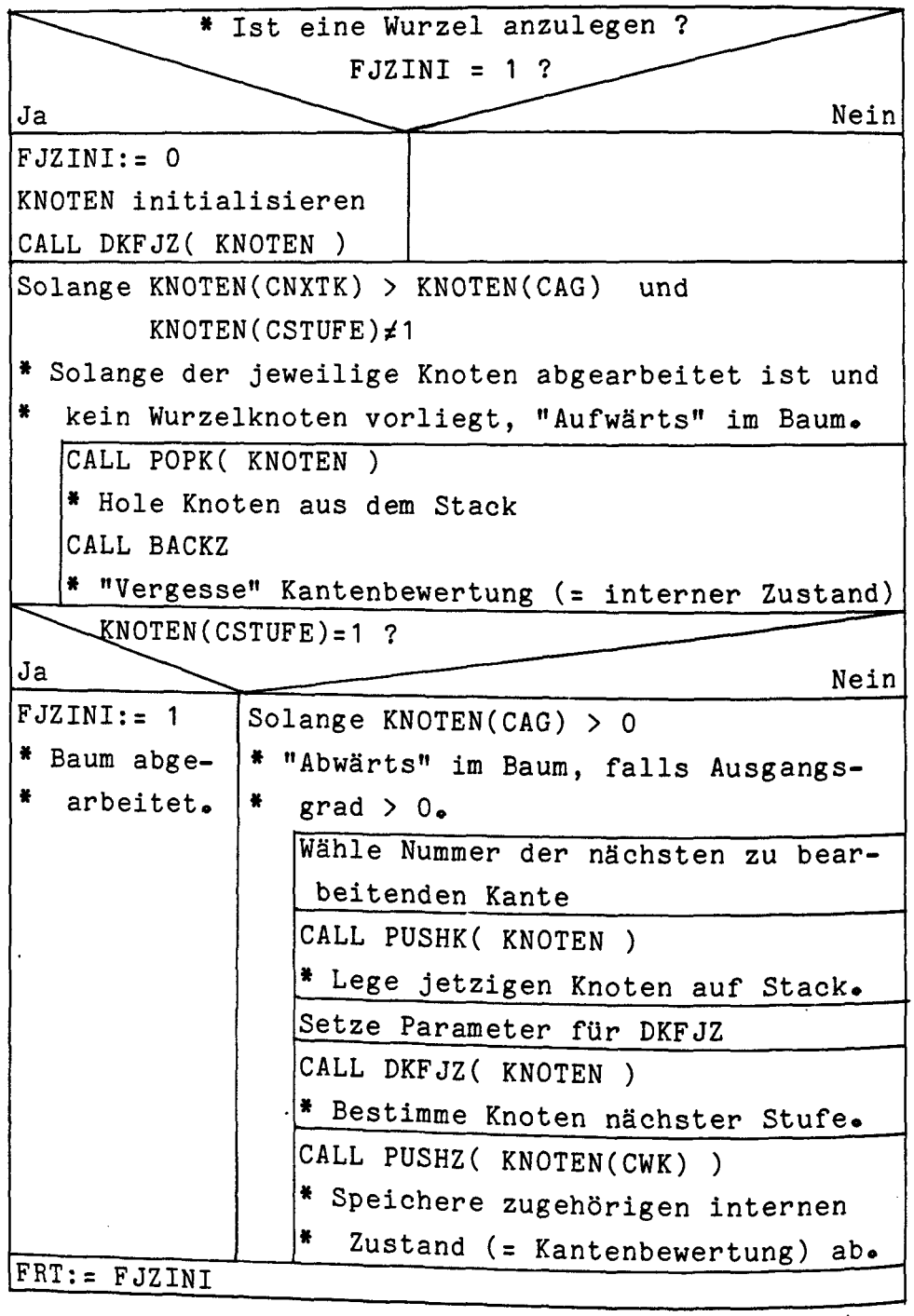
Anderenfalls wird in KNOTEN vermerkt, daß die nach Definition des FJ(z)-Baumes nächste Kante aufgesucht wird (KNOTEN(CNXTK) wird um 1 erhöht). KNOTEN wird in dieser Form auf den Stack gebracht (CALL PUSHK) und ein neuer Knoten definiert (CALL DKFJZ). Die Kantenbewertung wird im Stack abgelegt (CALL PUSHZ).

Ist der Ausgangsgrad des Knotens, KNOTEN(CAG), ungleich Null, wird dementsprechend eine weitere Kante aufgesucht. Sonst ist ein terminaler Knoten gefunden (FRT=0).

* Besonderheiten

Keine.

* Struktogramm



5.3.2.6 JNM

* JNM - Durchlaufe einen J(N,M)- oder G(N)-Baum.

* Aufruf: CALL JNM(N,M,INI,FRT,SET)

* Parameter

- Explizite Parameter

  - Eingabe

N	INTEGER, Zahl der Aufträge im betrachteten (Teil-) WS-Netz.
M	INTEGER, Zahl der Stationen im betrachteten (Teil-) WS-Netz.
INI	INTEGER. Für INI=1 wird eine Wurzel angelegt. Weitere Aufrufe mit INI=0 beziehen sich dann auf den durch die definierte Wurzel angelegten Baum, bis entweder der Baum abgearbeitet ist - dann wird der vorherige Baum bearbeitet, falls vorhanden - oder erneut ein Aufruf mit INI=1 erfolgt.
SET	INTEGER. Falls SET=CSET: Bewertungen der Kanten im Baum werden anhand des durchlaufenen Pfades mit PUSHZ/BACKZ im Stack abgespeichert. Ist SET=CNSET: keine Abspeicherung erfolgt. In diesem Fall handelt es sich um einen Aufruf zur Erzeugung eines G(N)-Baums. N bezeichnet dann die Zahl aller Aufträge im WS-Netz, M die Zahl der Teilsysteme, 2.

  - Ausgabe

FRT	INTEGER. Falls FRT=0: ein terminaler Knoten wurde gefunden, sonst: der Baum ist vollständig abgearbeitet.
INI	wird wie FRT gesetzt.

- Implizite Parameter

  Keine.

* Genutzte COMMON Bereiche

CKONST (Codierhilfe, bleibt unverändert).

* Funktionsablauf

Falls die Wurzel des J(N,M)-Baums anzulegen ist, werden die Parameter für DKJNM entsprechend gesetzt und DKJNM aufgerufen, die Variable KNOTEN damit initialisiert.

Sonst wird der terminale Knoten des letzten Aufrufs für diesen J(N,M)-Baum vom Stack geholt.

In beiden Fällen wird nun solange ein Knoten vom Stack geholt (CALL POPK) und, falls verlangt (SET=CSET), die

zugehörige Kantenbewertung eliminiert (CALL BACKZ), bis ein noch nicht vollständig abgearbeiteter Knoten gefunden wird oder der Stack leer ist (das ist gleichbedeutend mit einem abgearbeiteten Knoten der Stufe 1).

Bei leerem Stack wird FRT auf 1 gesetzt, was bedeutet, daß der J(N,M)-Baum vollständig durchlaufen worden ist.

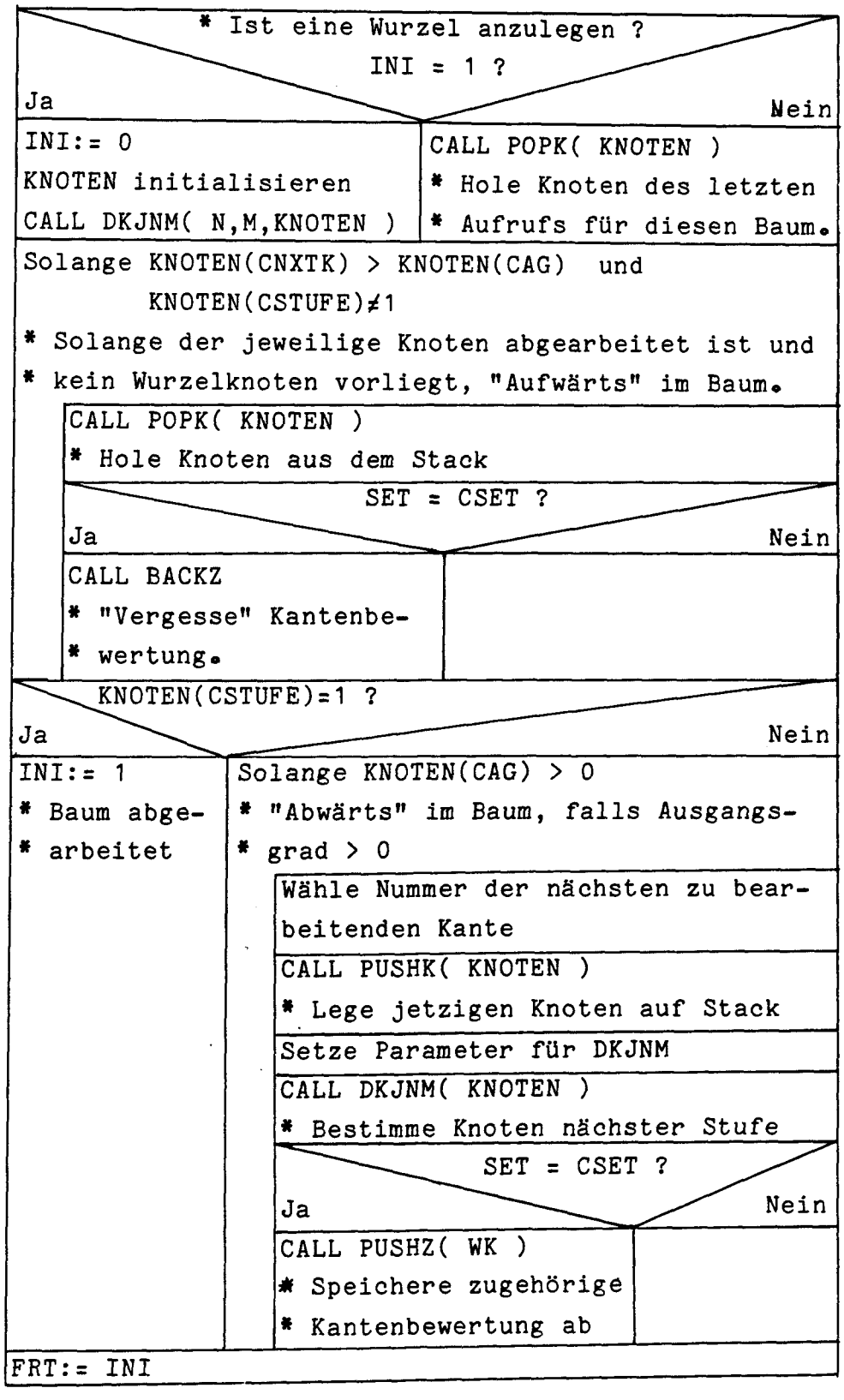
Anderenfalls wird in KNOTEN vermerkt, daß die nach Definition des J(N,M)-Baumes nächste Kante aufgesucht wird (KNOTEN(CNXTK) wird um 1 erhöht). KNOTEN wird in dieser Form auf den Stack gebracht (CALL PUSHK) und ein neuer Knoten definiert (CALL DKJNM). Die Kantenbewertung wird im Stack abgelegt (CALL PUSHZ).

Ist der Ausgangsgrad des Knotens, KNOTEN(CAG), ungleich Null, wird dementsprechend eine weitere Kante aufgesucht. Sonst ist ein terminaler Knoten gefunden (FRT=0).

* Besonderheiten

Keine.

* Struktogramm



5.3.2.7 DKFJZ

* DKFJZ - Initialisiere einen Knoten eines FJ(z)-Baums.

* Aufruf: CALL DKFJZ(KNOTEN)

* Parameter

- Explizite Parameter

  - Eingabe

    KNOTEN INTEGER, Dimension CDIMKN. Von Bedeutung sind für die Eingabe nur die Werte:

    KNOTEN(CSTUFE): Stufe des zu initialisierenden Knotens $K^{(i)}$.

    KNOTEN(CNDX): Index i des zu initialisierenden Knotens bezüglich des Vorgängers.

  - Ausgabe

    KNOTEN Von Bedeutung sind folgende Werte:

    KNOTEN(CSTUFE), KNOTEN(CNDX): unverändert.

    KNOTEN(CAG): Ausgangsgrad $AG(K^{(i)})$ des Knotens.

    KNOTEN(CWK): Wert $WK(K^{(i)})$.

    KNOTEN(CNXTK): 1. Ist der Index des Knotens nächster Stufe, der noch nicht von $K^{(i)}$ aus besucht wurde.

  - Implizite Parameter (nur Eingabe vorhanden)

    $Z INTEGER*2, COMMON $WRKSP, Maximaldimension $MAXZ, aktuelle Dimension $CURZ. Enthält in den ersten BM Komponenten die Auftragsverteilung im WS-Netz, also den Jacksonzustand.

    BM INTEGER*2, COMMON BFZS. Anzahl der Stationen im WS-Netz.

    BLI INTEGER*2, COMMON BFZS, Maximaldimension BMAXM, aktuelle Dimension BM. Enthält pro Station die Anzahl der internen aktiven Zustände.

* Genutzte COMMON Bereiche

BFZS
CKONST (Codierhilfe, bleibt unverändert).
$WRKSP

* Externe Routinen

Keine.

* Funktionsablauf

Entsprechend der Definition der Funktion AG des FJ(z)-Baums wird der Ausgangsgrad des Knotens bestimmt (KNOTEN(CAG)). Anschließend wird nach der Definition der Funktion WE für den FJ(z)-Baum der Wert der Kante vom Vorgänger zu diesem Knoten bestimmt (s. aber den nächsten Punkt).

* Besonderheiten

Abweichend von der angegebenen Funktionsdefinition wird für den Fall, daß kein Auftrag an der Station vorhanden ist, die Kantenbewertung zu Null gesetzt.

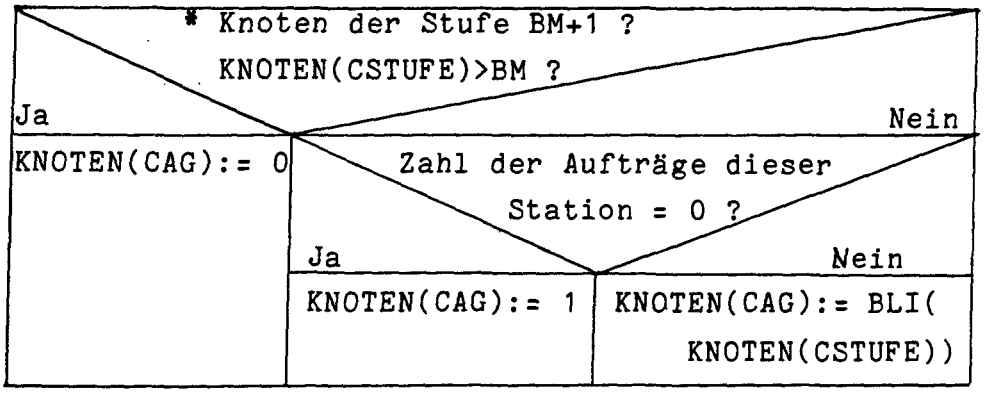
Wie in der Definition erwähnt, spielt die Bewertung dieser einzigen, zu dem bearbeiteten Knoten führenden Kante für das Durchlaufen des Baumes keine Rolle.

Andererseits bleibt dann jede Kantenbewertung als interner Zustand einer Station interpretierbar.

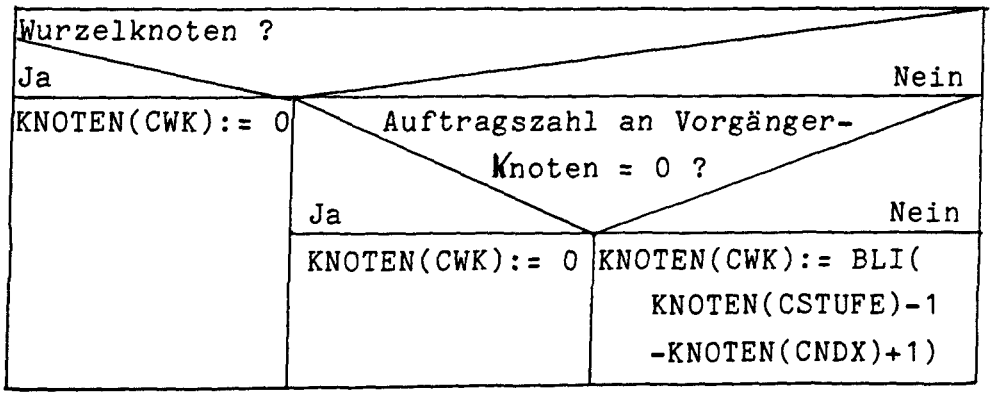
* Struktogramm

Ausgangsgrad bestimmen
Kantenbewertung berechnen
KNOTEN(CNXTK):= 1

Ausgangsgrad bestimmen: KNOTEN(CAG)



Kantenbewertung berechnen: KNOTEN(CWK)



5.3.2.8 DKJNM

* DKJNM - Initialisiere einen Knoten eines J(N,M)-Baums.

* Aufruf: CALL DKJNM(N,M,KNOTEN)

* Parameter

- Explizite Parameter

  - Eingabe

    N INTEGER, Anzahl der Aufträge im betrachteten (Teil-) WS-Netz.

    M INTEGER, Zahl der Stationen im betrachteten (Teil-) WS-Netz.

    KNOTEN INTEGER, Dimension CDIMKN. Für die Eingabe Wichtig sind:

    KNOTEN(CSTUFE): Stufe des zu initialisierenden Knotens $K^{(i)}$.

    KNOTEN(CNDX): Index i des zu initialisierenden Knotens $K^{(i)}$.

    KNOTEN(CWK): Wert WK(K) des Vorgängerknotens K.

    KNOTEN(CAG): Ausgangsgrad AG(K) des Vorgän- gerknotens K.

  - Ausgabe

    KNOTEN Folgende Werte sind gesetzt:

    KNOTEN(CSTUFE), KNOTEN(CNDX): unverändert.

    KNOTEN(CWK): Wert des Knotens $K^{(i)}$.

    KNOTEN(CAG): Ausgangsgrad $AG(K^{(i)})$.

    KNOTEN(CNXTK): 1, Index des von $K^{(i)}$ aus als nächsten zu besuchenden Nachfolger.

- Implizite Parameter

  Keine.

* Genutzte COMMON Bereiche

CKONST (Codierhilfe, bleibt unverändert).

* Externe Routinen

Keine.

* Funktionsablauf

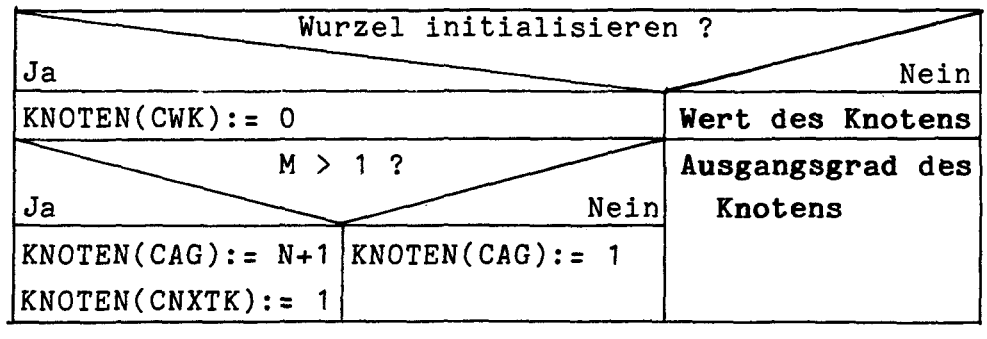
Entsprechend der Definition des Wertes WK eines Knotens eines J(N,M)-Baums wird der Wert des Knotens bestimmt. Anhand der Definition der Funktion AG eines J(N,M)-Baums wird der Ausgangsgrad berechnet.

Abschließend wird KNOTEN(CNXTK) auf 1 gesetzt.

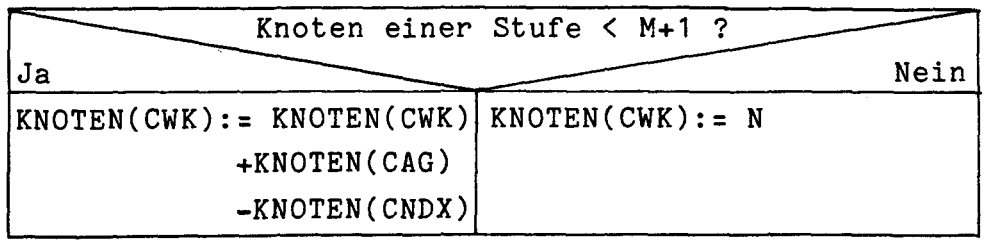
* Besonderheiten

Für den Wurzelknoten ist als Eingabe nur die Spezifikation von KNOTEN(CSTUFE)=1 erforderlich.

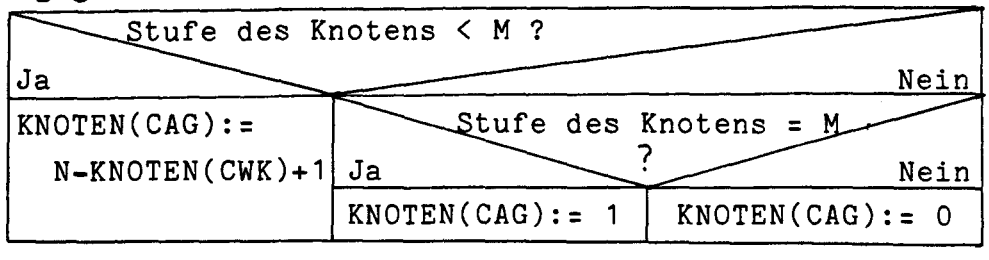
* Struktogramm



Wert des Knotens: KNOTEN(CWK)



Ausgangsgrad des Knotens: KNOTEN(CAG)



5.3.2.9 PUSHK

* PUSHK - Abspeichern eines Knotens im Stack.

* Aufruf: CALL PUSHK(KNOTEN)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  KNOTEN INTEGER, Dimension CDIMKN, ist der abzuspeichernde Knoten.

- Implizite Parameter (Ein- und Ausgabe)

  $STCKK INTEGER*2, COMMON $WRKSP, Dimension $MAXSK. Vektor, der zur Simulation eines Stacks dient. KNOTEN wird dahin übertragen.
  $MAXSK INTEGER*2, COMMON $WRKSP. Maximaldimension von $STCKK. Bleibt unverändert und dient nur der Überlaufkontrolle.
  $CURSK INTEGER*2, COMMON $WRKSP. Enthält Index des letzten belegten Elements in $STCKK.

* Genutzte COMMON Bereiche

CKONST (Codierhilfe, bleibt unverändert).
$WRKSP

* Externe Routinen

Keine.

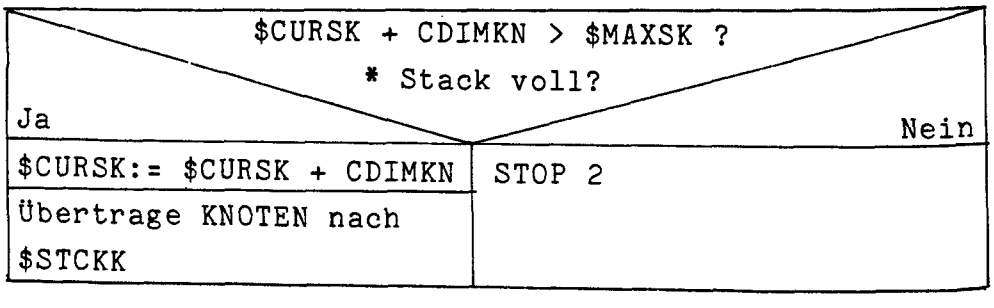
* Funktionsablauf

Die Elemente von KNOTEN werden ab dem Index 1 bis CDIMKN nach $STCKK übertragen (ab Index $CURSK + 1), $CURSK entsprechend erhöht.

* Besonderheiten

Bei Überlauf des Stacks: STOP 2.

* Struktogramm



5.3.2.10 POPK

* POPK - Hole den obersten Knoten vom Stack.

* Aufruf: CALL POPK(KNOTEN)

* Parameter

- Explizite Parameter (nur Ausgabe vorhanden)

  KNOTEN INTEGER, Dimension CDIMKN. Enthält nach dem Aufruf die Werte des obersten Knotens, der zum Zeitpunkt des Aufrufs im Stack war.

- Implizite Parameter (Ein- und Ausgabe)

  $STCKK INTEGER*2, COMMON $WRKSP, Maximaldimension $MAXSK. Enthält alle bisher über PUSHK in den Stack gebrachten Knoten.
  $CURSK INTEGER*2, COMMON $WRKSP. Index des letzten Elements des obersten Knotens im Stack. Enthält nach Aufruf einen Wert gleicher Bedeutung, der jedoch um CDIMKN kleiner ist.

* Genutzte COMMON Bereiche

CKONST (Codierhilfe, bleibt unverändert).
$WRKSP

* Externe Routinen

Keine.

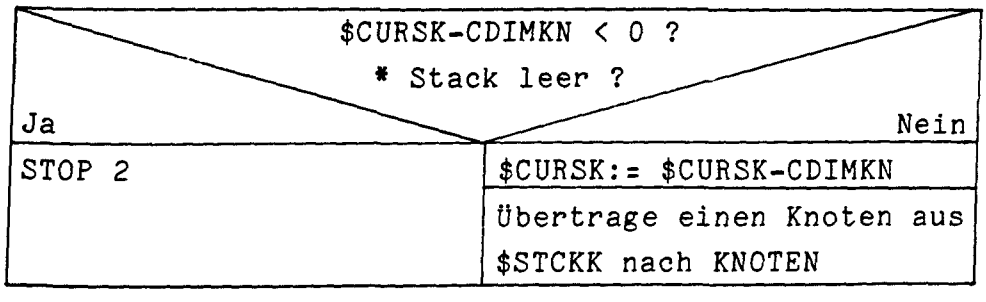
* Funktionsablauf

Die Elemente des obersten Knotens im Stack $STCKK werden von $CURSK-CDIMKN+1 bis $CURSK in die entsprechenden Komponenten von KNOTEN übertragen, $CURSK um CDIMKN vermindert.

* Besonderheiten

Falls bei Aufruf der Stack schon leer ist: STOP 2.

* Struktogramm



5.3.2.11 PUSHZ

* PUSHZ - Abspeichern eines Wertes auf Stack.

* Aufruf: CALL PUSHZ(ZI)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  ZI INTEGER. Abzuspeichernder Wert, muß kleiner als 32768 sein.

- Implizite Parameter (Ein- und Ausgabe)

  $Z INTEGER*2, Maximaldimension $MAXZ. Stellt den Stack dar.
  $MAXZ INTEGER*2, gibt die Maximalzahl der Elemente in $Z an.
  $CURZ INTEGER*2, Index des ersten freien Elements in $Z.

* Genutzte COMMON Bereiche

$WRKSP

* Externe Routinen

Keine.

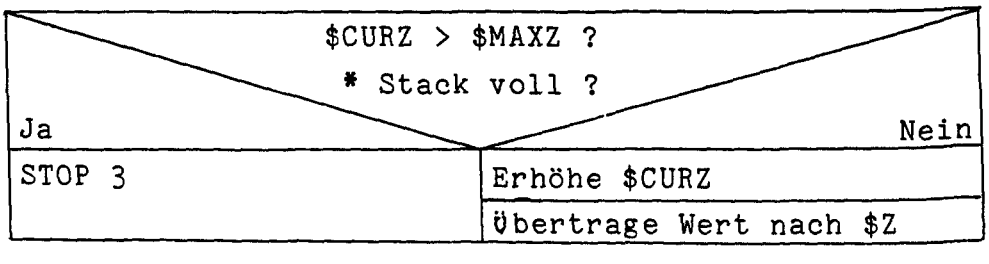
* Funktionsablauf

Der Wert wird an die durch $CURZ angegebene Stelle gebracht und $CURZ entsprechend erhöht.

* Besonderheiten

Falls der Stack überläuft: STOP 3.

* Struktogramm



5.3.2.12 BACKZ

* BACKZ - Entferne obersten Eintrag aus dem Stack.

* Aufruf: CALL BACKZ

* Parameter

- Explizite Parameter

  Keine.

- Implizite Parameter (Ein- und Ausgabe)

$Z	INTEGER*2, Maximaldimension $MAXZ. Stellt den Stack dar.
$MAXZ	INTEGER*2, gibt die Maximalzahl der Elemente in $Z an.
$CURZ	INTEGER*2, Index des ersten freien Elements in $Z.

* Genutzte COMMON Bereiche

$WRKSP

* Externe Routinen

Keine.

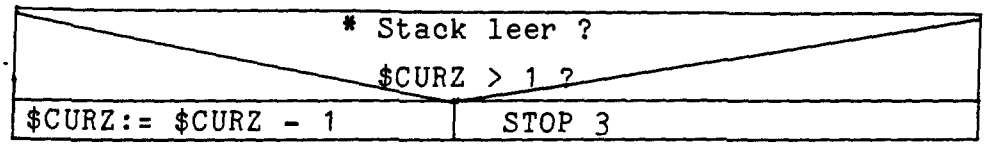
* Funktionsablauf

$CURZ wird um 1 vermindert.

* Besonderheiten

Falls der Stack bei Aufruf leer ist: STOP 3.

* Struktogramm



5.3.3 GNTRNS UND ABHÄNGIGE ROUTINEN

5.3.3.1 GNTRNS

* GNTRNS - Generiere Transitionen des WS-Netzes.

* Aufruf: CALL GNTRNS(CURZ,EPS)

* Parameter

- Explizite Parameter (Ein- und Ausgabe)

  CURZ INTEGER*2, Dimension 2*BM. Arbeitsbereich, wird deshalb als Parameter übergeben, um Änderungen der Dimensionierung ohne Neukompilierung vornehmen zu können.

  EPS REAL. Rückgabewert. Minimum aller positiven Matrixelemente. Dient später zu Genauigkeitsabschätzungen und muß initialisiert sein.

- Implizite Parameter (nur Eingabe vorhanden)

  BM INTEGER*2, COMMON BFZS, Anzahl der Stationen im WS-Netz.

  BLI INTEGER*2, COMMON BFZS. Vektor der Maximallänge BMAXM, der aktuellen Länge BM. Enthält pro Station i die Anzahl l_i der internen aktiven Zustände.

  BZS INTEGER*2, COMMON BFZS. Vektor der aktuellen Länge ANZ*BM*2. Entspricht Z(S) mit der bei der Definition der Partition π angegebenen Ordnung.

  AJZ INTEGER, COMMON AFZS, Dimension maximal AMAXJZ, aktuell ANJZ. Enthält zu den Werten in API analoge für Jacksonzustände statt für Grobzustände, d. h. jeweils den Index des ersten zugehörigen Feinzustands.

  ANJZ INTEGER, COMMON AFZS, Bedeutung s. AJZ.

* Genutzte COMMON Bereiche

AFZS
BFZS
GKONST (Codierhilfe, bleibt unverändert.)

* Externe Routinen

- SRCHZS: INTEGER FUNCTION, suche den Index eines gegebenen Feinzustands.
- ESTORE: Abschlußarbeiten für die Abspeicherung der Matrixelemente.
- GNRATE: Berechne Übergangsraten und leite Abspeicherung ein.

* Funktionsablauf

Alle möglichen Übergänge sind zu erzeugen. Es gibt zwei Arten von Zustandsübergängen, exogene, das sind solche mit Wechsel des Jacksonzustands, und endogene, solche mit gleichem Jacksonzustand.

Pro Jacksonzustand wird jeder Feinzustand als Ausgangspunkt genommen. LJZ indiziert den gerade betrachteten Jacksonzustand, AJZ(LJZ) gibt den Index des ersten Feinzustands in diesem Jacksonzustand wieder. Aufgrund der in Z(S) definierten Ordnung liegen alle Indizes der Feinzustände eines Jacksonzustands in einem Intervall, ohne solche anderer Feinzustände dazwischen.

Die Auftragsverteilung des Jacksonzustands wird in den Arbeitsbereich übertragen.

Ausgehend von jedem Feinzustand des Jacksonzustands werden alle endogenen Übergänge innerhalb dieses Jacksonzustands erzeugt.

Dazu werden die internen Zustände des Ausgangsfeinzustands in den Arbeitsbereich übertragen. Für jede Station, die sich nicht im Ruhezustand befindet, wird ihr interner Zustand mit jedem ihrer anderen, aktiven, internen Zustände kombiniert. Die Funktion SRCHZS sucht dann den Index des Zielzustands, GNRATE berechnet die Rate und leitet die Abspeicherung des entsprechenden Matrixelementes ein.

Nach Abschluß der Erzeugung aller endogenen Übergänge werden nun die exogenen berechnet, immer noch vom selben Jacksonzustand ausgehend.

Pro Station ist jetzt folgendes zu tun:

Die Auftragszahl der Station ist, falls möglich, um 1 zu vermindern (wenn nicht möglich, d. h. es sind schon keine Aufträge mehr an der Station, wird die nächste Station betrachtet). Pro jetzt noch möglichem internen Folgezustand der Station (das kann der Ruhezustand sein, falls die Auftragszahl an der betrachteten Station nun Null ist, oder sonst alle aktiven internen Zustände) ist diese Änderung jeweils mit entsprechenden Änderungen an allen anderen Stationen zu verbinden. Das bedeutet: für jede andere Station ist deren Auftragszahl um 1 zu erhöhen und die restliche Auftragsverteilung beizubehalten. Für jene Station ist dann jeder mögliche interne Zustand zu betrachten. Falls die Auftragszahl schon vor der Erhöhung positiv war, ist der dort vorhanden gewesene interne Zustand der einzig mögliche (interne) Folgezustand der Station, sonst alle aktiven internen Zustände.

Die internen Zustände aller übrigen, von der Auftragsumverteilung nicht berührten Stationen im Zielfeinzustand müssen ebenfalls noch bestimmt werden. Diese sind aber dieselben wie im Ausgangsfeinzustand.

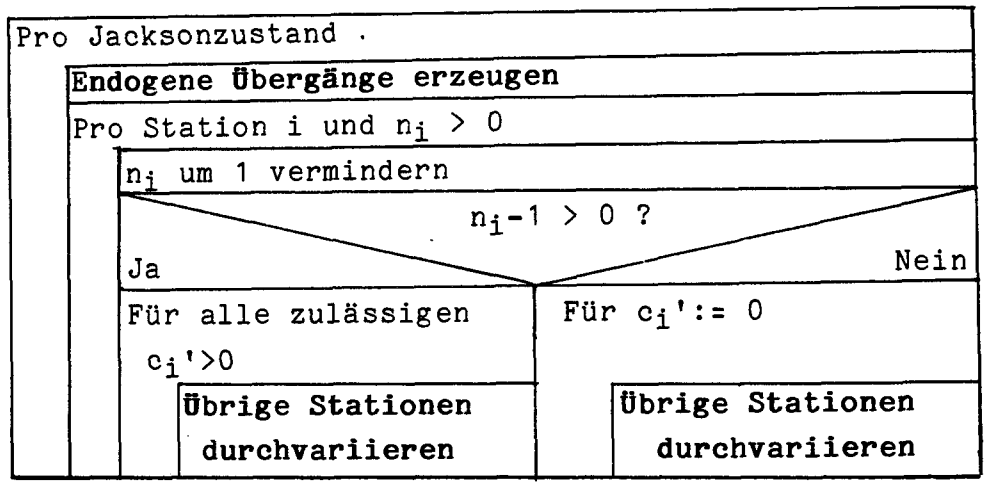
Der sich aufgrund dieser Vorgehensweise ergebende Zielfeinzustand ist Parameter für SRCHZS. Der gelieferte Index sowie alle übrigen notwendigen Werte werden an GNRATE übergeben.

Sind alle Jacksonzustände abgearbeitet, so wird ESTORE zur Erledigung der Abschlußarbeiten für die Abspeicherung der Matrixelemente aufgerufen.

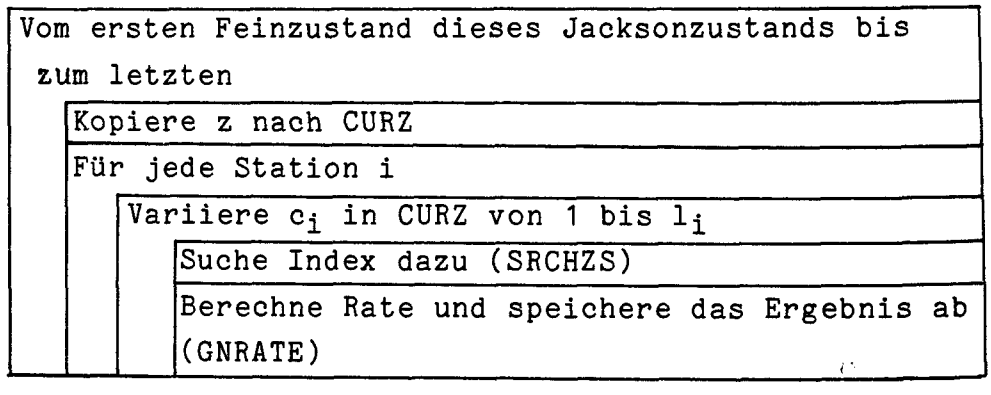
* Besonderheiten

  Keine.

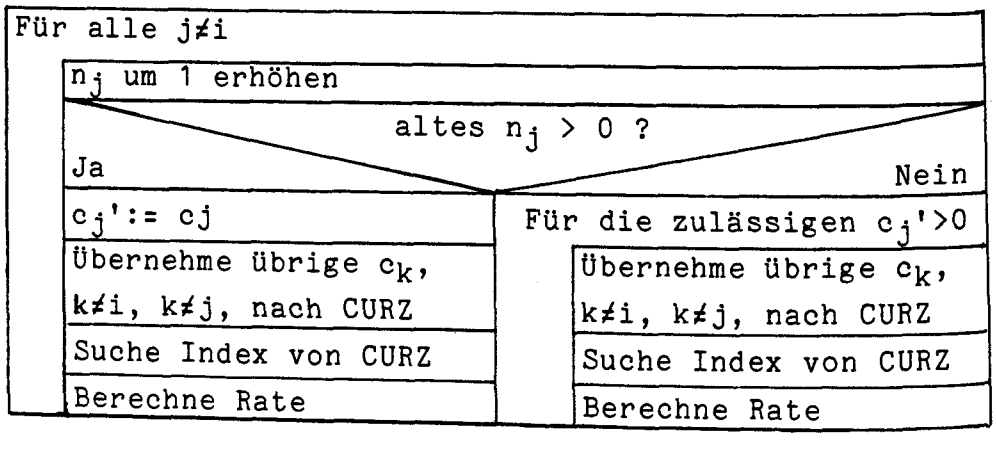
* Struktogramm



Endogene Übergänge erzeugen



Übrige Stationen durchvariieren



5.3.3.2 SRCHZS

* SRCHZS - INTEGER FUNCTION, suche den Index eines Feinzustands.

* Aufruf: I= SRCHZS(CURZ,IZA,IZE,TYP)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

CURZ	INTEGER*2, Dimension BM2. Feinzustand, dessen Index gesucht ist.
IZA	INTEGER. Anfangsindex bezüglich BZS für die Suche, falls TYP=GJACK. Wird sonst nicht verwendet.
IZE	INTEGER. Endindex bezüglich BZS für die Suche, falls TYP=GGROB. Wird sonst nicht verwendet.
TYP	INTEGER*2. Spezifiziert die Art und Größe des Suchbereichs. TYP=GJACK: Suchbereich ist von IZA bis IZE, d. h. innerhalb eines Jacksonzustands. TYP=GGROB: der Suchbereich umfaßt den Grobzustand, in dem der in CURZ angegebene Feinzustand liegt.

- Implizite Parameter (nur Eingabe vorhanden)

BM	INTEGER*2, COMMON BFZS. Anzahl der Stationen im WS-Netz.
BM2	INTEGER*2, COMMON BFZS. Ist gleich 2*BM, d. h. Anzahl der Komponenten eines Feinzustands.
BMMNS1	INTEGER*2, COMMON BFZS. Ist gleich BM-1, Hilfsgröße.
BMPLS1	INTEGER*2, COMMON BFZS. Ist gleich BM+1, Hilfsgröße.
BKMNS1	INTEGER*2, COMMON BFZS, ist gleich BK-1, d. h. Nummer der letzten Station im Teilsystem S_1.
BN	INTEGER*2, COMMON BFZS. Anzahl der Aufträge im WS-Netz.
BZS	INTEGER*2, COMMON BFZS, enthält alle Zustände von Z(S) in der durch fz angegebenen Reihenfolge.
API	INTEGER, COMMON AFZS, Partition π in modifizierter Form (s. Abschnitt 5.2.2).

* Genutzte COMMON Bereiche

AFZS
BFZS
GKONST (Codierhilfe, bleibt unverändert.)

* Externe Routinen

Keine.

* Funktionsablauf

SRCHZS führt eine binäre Suche im angegebenen Bereich durch.

Falls nur innerhalb eines Jacksonzustands gesucht werden soll, werden die unteren und oberen Indexgrenzen entsprechend den Werten von IZA und IZE gesetzt.

Sonst wird anhand der ersten BK-1 Elemente von CURZ der Grobzustand berechnet, in dem der durch CURZ angegebene Feinzustand liegt. Mit Hilfe von API wird die untere und obere Indexgrenze bestimmt.

Für beide Fälle wird die Intervallmitte berechnet. Der dort liegende Feinzustand wird mit dem in CURZ verglichen. Und zwar werden, falls der Suchbereich nur ein Jacksonzustand ist, ausschließlich die internen Zustände miteinander verglichen, sonst auch noch die Auftragsverteilung.

Ist eine Komponente des Feinzustands in der Intervallmitte kleiner als die entsprechende in CURZ, so hat der Feinzustand in CURZ lexikographisch einen höheren Wert, also einen niedrigeren Index. Dementsprechend ist die obere Indexgrenze zu senken. Ist eine Komponente dagegen größer, gilt analog: die untere Indexgrenze ist anzuheben.

Bei Übereinstimmung wird der Indexwert der Intervallmitte bezüglich Z(S) zurückgegeben.

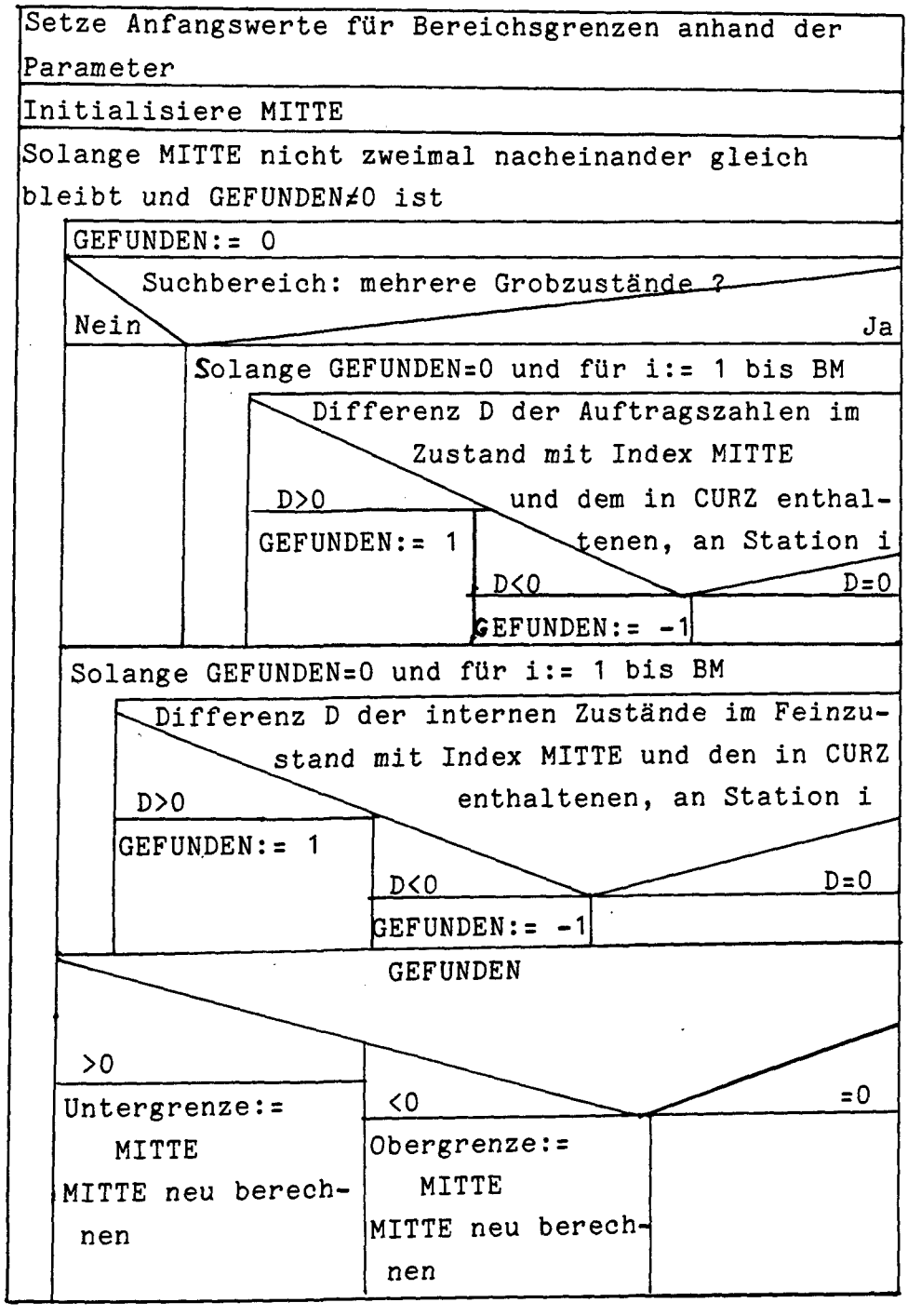
* Besonderheiten

Bei Angabe des Suchbereichs "Jacksonzustand" wird nicht überprüft, ob die angegebenen Indexwerte tatsächlich zu ein- und demselben Jacksonzustand gehören.

Der angegebene Feinzustand wird nicht auf Gültigkeit überprüft, d. h. nicht daraufhin, ob die angegebene Auftragsverteilung und die internen Zustände möglich sind.

Folge einer fehlerhaften Angabe ist Nichtauffinden des Zustands, dies führt zu STOP 20.

* Struktogramm



5.3.3.3 GNRATE

* GNRATE - Generiere Übergangsrate.

* Aufruf: CALL GNRATE(TYP,IZS,FZI,FZSJ,I,J,CI,CIS,CJ,CJS,EPS)

* Parameter

- Explizite Parameter

  - Eingabe

    TYP — INTEGER*2, ist ein Code für die Art des Übergangs, dessen Rate zu berechnen ist.

    TYP=GENDO: endogener Übergang,

    TYP=GEXO: exogener Übergang.

    IZS — INTEGER, Index bezüglich BZS für den Ausgangszustand z.
    FZI — INTEGER, Index fz(z) des Ausgangszustands z.
    FZSJ — INTEGER, Index fz(z') des Folgezustands z'.
    I — INTEGER, bezeichnet die Station i.
    J — INTEGER, bezeichnet die Station j.
    CI — INTEGER, interner Ausgangszustand c_i der Station i.
    CIS — INTEGER, interner Folgezustand c_i' der Station i.
    CJ — INTEGER, interner Ausgangszustand c_j der Station j.
    CJS — INTEGER, interner Folgezustand c_j' der Station j.

  - Ausgabe

    EPS — REAL. Ist, bei Rückgabe, das bisherige Minimum aller positiven Übergangsraten (=Matrixelemente).

- Implizite Parameter (nur Eingabe vorhanden)

  BLI — INTEGER*2, COMMON BFZS, Maximaldimension BMAXM, aktuelle Dimension BM. BLI(m) enthält die Anzahl l_m an aktiven Zuständen der Station m.
  WST — REAL, COMMON WSTMY, Maximaldimension (BMAXM,BMAXLI+2,BMAXLI+2). Für jede Station m, $m \in 1..BM$, enthält WST in den Komponenten (m, 1, 1) bis (m, BLI(m)+2, BLI(m)+2) die zugehörige Typmatrix.
  WSW — REAL, COMMON WSTMY, Maximaldimension (BMAXM,BMAXM), aktuelle Dimension (BM,BM). Enthält die Wechselmatrix W der Ordnung (M,M).

* Genutzte COMMON Bereiche

BFZS
WSTMY

* Externe Routinen

- MY: REAL FUNCTION, bestimme Bedienrate μ einer Station.
- QSTORE: Speichere ein Matrixelement ab.

* Funktionsablauf

Je nach Aufruftyp wird die entsprechende Übergangsrate berechnet. Nur bei Raten ungleich Null wird EPS neu bestimmt und QSTORE zum Abspeichern aufgerufen.

* Besonderheiten

Falls der Aufruftyp unbekannt ist, STOP 6.

* Struktogramm

Endogene Rate berechnen ?	
Ja	Nein
Berechne Rate anhand der Formel (Abschnitt 4.4)	Berechne exogene Rate anhand der Formel (Abschnitt 4.4)
Rate = 0 ?	
Ja	Nein
	* Matrixelement abspeichern CALL QSTORE

5.3.3.4 MY

* MY - REAL FUNCTION, bestimme die Bedienrate μ einer Station.

* Aufruf: R= MY(STAT,IZS)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

STAT	INTEGER, Station i, deren Bedienrate $\mu^{(i)}(z)$ geliefert werden soll.
IZS	INTEGER, Index des Zustandes z, bezüglich BZS.

- Implizite Parameter (nur Eingabe vorhanden)

  MYTP INTEGER*2, Maximaldimension BMAXM, aktuelle Dimension BM. Gibt pro Station i die Art der Abhängigkeit der Bedienrate vom Zustand z des WS-Netzes an:

  MYTP(i)=

  KCI, Bedienrate abhängig vom internen Zustand c_i.

  KFNI, Bedienrate abhängig von der Anzahl n_i der Aufträge an der Station, und zwar geht n_i als Faktor ein (Bedienstrategie IS).

  KDNI, Bedienrate abhängig von der Anzahl n_i der Aufträge an der Station, und zwar geht n_i als Divisor ein (Bedienstrategie PS).

  KON, Bedienrate konstant, also unabhängig von z.

  BMMNS1 INTEGER*2, COMMON BFZS, ist gleich BM-1 und dient zum Auffinden des internen Zustands der Station STAT in BZS.

  BZS INTEGER*2, COMMON BFZS, enthält alle Zustände von Z(S) in der durch fz angegebenen Reihenfolge.

  WSMY REAL, COMMON WSTMY, Maximaldimension (BMAXM,BMAXLI). Enthält pro Station ihre Bedienrate oder zumindest einen Faktor davon, je nach Spezifikation in MYTP.

* Genutzte COMMON Bereiche

BFZS
KONST (Codierhilfe, bleibt unverändert.)
MYCOM
WSTMY

* Externe Routinen

Keine.

* Funktionsablauf

Die Art der Abhängigkeit der Bedienrate vom Zustand des WS-Netzes wird festgestellt.

Ist die Rate vom internen Zustand c_i der Station i=STAT abhängig (MYTP(i)=KCI), so liefert MY den Wert WSMY(i, c_i).

Sollte die Rate konstant sein (MYTP(i)=KON), wird der Wert WSMY(i, 1) zurückgegeben, sonst das Produkt oder der Quotient aus WSMY(i, 1) und n_i, je nachdem, ob MYTP(i)=KFNI oder MYTP(i)=KDNI.

* Besonderheiten

Die Gültigkeit der Codes in MYTP wird nicht überprüft.

* Struktogramm

NDXG:= MYTP(STAT) * Stelle Art der Abhängigkeit der Bedienrate vom * Systemzustand fest
Je nach Übereinstimmung mit einer der Größen KCI, KFNI, KDNI oder KON ist die Rate zu bestimmen

5.3.3.5 QSTORE

* QSTORE - Speichere ein Matrixelement ab.

* Aufruf: CALL QSTORE(J,I,QEL)

* Parameter (s. auch "Funktionsablauf")

- Explizite Parameter (nur Eingabe vorhanden)

J	INTEGER, Zeilenindex j der Matrix Q_π^T.
I	INTEGER, Spaltenindex i der Matrix Q_π^T.
QEL	REAL, abzuspeicherndes Element q_{ji} der Matrix Q_π^T.

- Implizite Parameter (Ein- und Ausgabe)

BN	INTEGER*2, COMMON BFZS, Anzahl der Aufträge im WS-Netz.
BNPLS1	INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.
API	INTEGER, COMMON AFZS, Partition π in modifizierter Form (s. Abschnitt 5.2.2).
ANZ	INTEGER, COMMON AFZS. Anzahl n der Zustände des WS-Netzes.
$LMAX	INTEGER*2, COMMON $LST, Maximalzahl der Elemente in der Freiplatzliste.
$FRMIN	INTEGER*2, COMMON $LST, Minimalzahl der Freiplatzelemente in $LISTE während der Ausführung, Rückgabewert.

* Genutzte COMMON Bereiche

AFZS	
BFZS	
DLST	
KONST	(Codierhilfe, bleibt unverändert.)
Q	
QLST	
$LST	

* Externe Routinen

  - ALLOC: INTEGER FUNCTION, ordne ein freies Listenelement zu.

  - QSTQT: Übertrage eine Blockdiagonalmatrix von $LISTE nach QT.

  - QSTQTE: Übertrage fertige Blocknebendiagonalmatrizen von $LISTE nach QTE.

* Funktionsablauf

  - Arbeitsbereiche

    QSTORE verwendet die Arbeitsbereiche QT, AQT, QTE, AQTE, AQTPJ, AQTEPJ, QSKU, QSKO, QRQ, $LISTE und QLISTE. Einen Überblick über Zusammenhänge und Bedeutung dieser Variablen liefert der Abschnitt 5.2.2.

    Allein hier beschriebene Arbeitsbereiche sind ANDXGI, DQT und DQTE.

  - Bedingungen an die Reihenfolge der Aufrufe

    Es ist für die Funktion von QSTORE erforderlich, daß die Folge der Werte von I monoton wächst. Außerdem: wenn I der Index eines Feinzustands des Grobzustands GI ist, so darf J nur aus GI oder den benachbarten Grobzuständen sein.

  - Zur Wahl der Listenstruktur

    Für die Abspeicherung der Elemente von Q_π^T ist die Einhaltung der aufsteigenden Ordnung der Spaltenindizes innerhalb einer Zeile erforderlich, weil die angewendeten numerischen Verfahren dies verlangen.

    Die Zustandsübergänge werden auch in steigender Reihenfolge der Spaltenindizes erzeugt, nur sind die Zeilenindizes zweier nacheinander erzeugter Übergänge nicht unbedingt aufsteigend. Bei der gewählten sequentiellen Abspeicherung aller Elemente einer Zeile (in QT bzw. QTE) würde fast jeder Zeilenwechsel ein Verschieben der Einträge in den höherwertigen Zeilen erfordern. Zur Umgehung dieses Sachverhalts wurde eine temporäre Listenstruktur gewählt. Temporär deshalb, weil nach Abschluß eines Grobzustands, d. h. wenn der Spaltenindex in den Indexbereich eines neuen Grobzustands wechselt, einige Blockuntermatrizen von Q_π^T vollständig sind, also aus den Listen nach QT und QTE übertragen werden können.

  - Ablauf

    Beim ersten Aufruf von QSTORE überhaupt oder nach vorausgehender Ausführung von ESTORE werden alle Arbeitsbereiche, soweit nötig, initialisiert: QSKU, QSKO und QRQ werden zu 0.0 gesetzt, AQTPJ und AQTEPJ auf KNULL, die LOGICAL-Variablen DQT und DQTE auf .FALSE. . Die Freiplatzliste wird angelegt. Der Vektor ANDXGI wird so initialisiert, daß er zu jedem Index von Z(S) den Index des zugehörigen Grobzustands liefert.

Im bei jedem Aufruf ausgeführten Teil wird der Index GI des zum Spaltenindex I gehörigen Grobzustands ermittelt.[8] Liegt ein Grobzustandswechsel vor, d. h. war I beim letzten Aufruf noch aus dem Grobzustand mit Index GI-1, dann sind jetzt die Blockdiagonaluntermatrix $Q^T_{GI-1,GI-1}$ und alle Zeilen aus den Grobzustandsbereichen mit einem Index kleiner als GI-1 vollständig.

Der Wechsel des Grobzustands wird in DQT vermerkt. Für die Speicherbelegungsstatistik wird das Minimum an freiem Platz in der Liste bestimmt. Jetzt wird QSTQT aufgerufen, um alle Elemente der Matrix $Q^T_{GI-1,GI-1}$ aus der Liste nach QT zu übertragen und die Listenelemente freizugeben.

Analog wird mit DQTE und QSTQTE für die in QTE abzuspeichernden Matrizen $Q^T_{GI-2,GI-3}$ und $Q^T_{GI-2,GI-1}$ verfahren.

In jedem Fall bleibt noch das neue Matrixelement, QEL, in die Liste einzutragen. Dazu wird von ALLOC der Index eines freien Listenelements beschafft und anhand von J bestimmt, ob QEL aus der unteren oder oberen Blocknebendiagonalen, oder aus der Blockdiagonalen ist. Entsprechend werden in QSKU, QSKO oder QRQ die Spaltensummen (Index I) gebildet. Wenn QEL nicht zur Blockdiagonalen gehört, wird die Liste, deren Endindex in AQTEPJ(KPJCUR,J) steht, gewählt. Anderenfalls wird stattdessen AQTPJ(KPJCUR,J) als Endpunkt der Liste betrachtet.

QEL wird als neues letztes Element in die Liste eingetragen.

* Besonderheiten

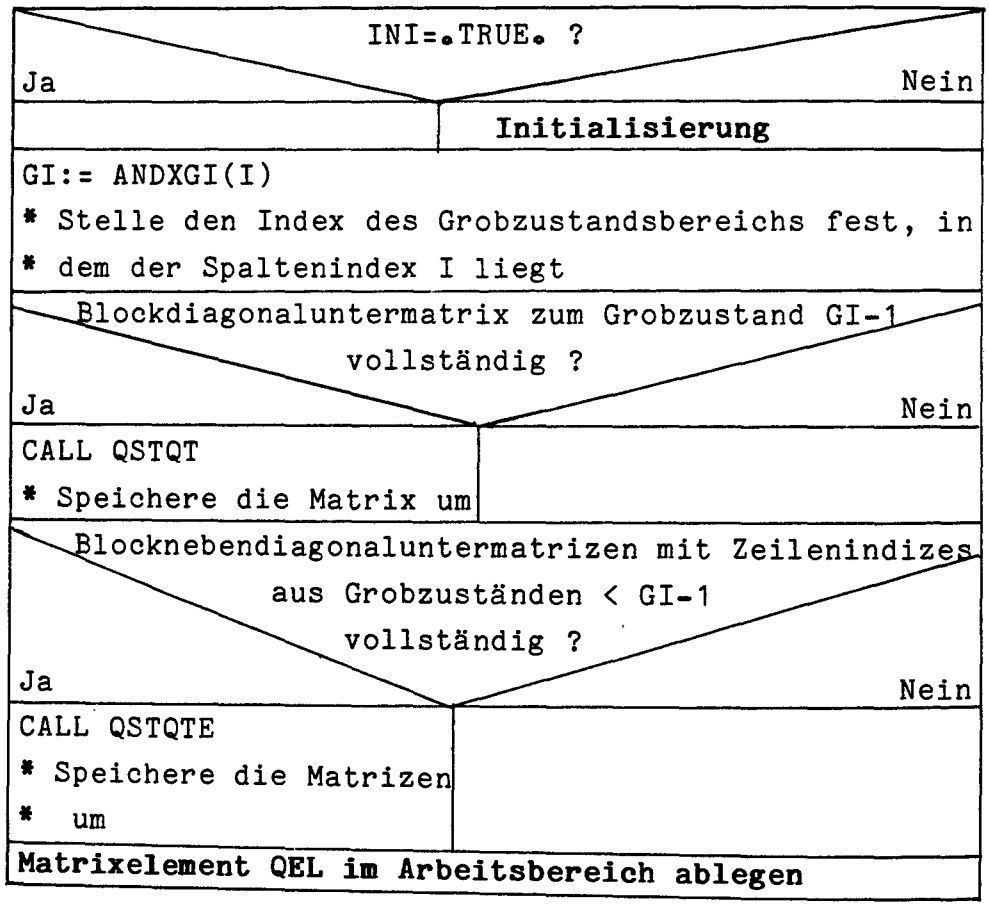
QSTORE enthält ESTORE als ENTRY.

Falls der zu J gehörige Grobzustand nicht derselbe wie der von I oder nicht zu diesem benachbart ist, STOP 54 (d. h. Q^T_π ist nicht blocktridiagonal).

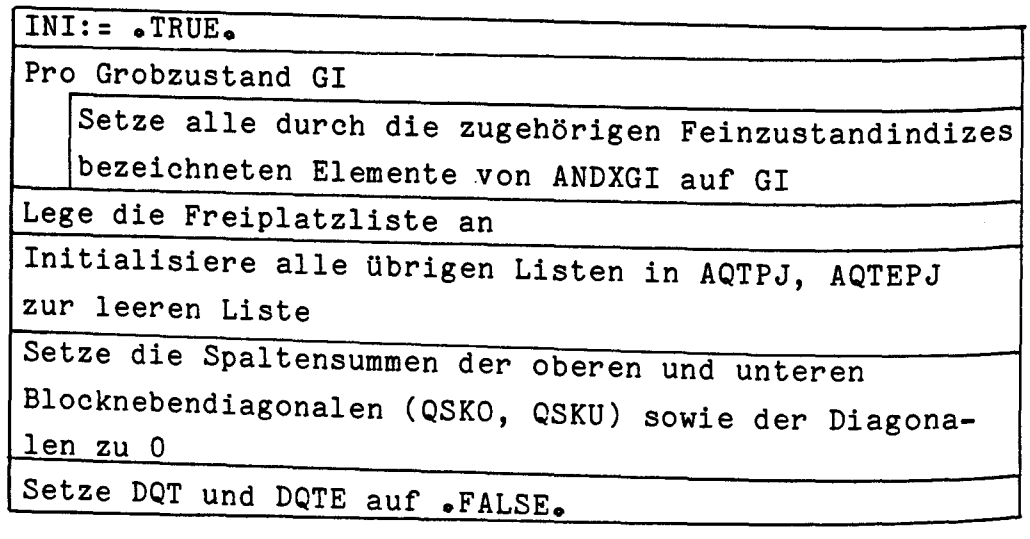
---

[8] Weil in FORTRAN IV Indizes nur positiv sein dürfen, bezeichnet GI immer den um 1 erhöhten Grobzustandsindex. Bei allen Rechnungen dieses Abschnitts, die mit GI durchgeführt werden, ist deshalb das Ergebnis jeweils noch um 1 zu vermindern, um den wahren Grobzustandsindex zu erhalten.

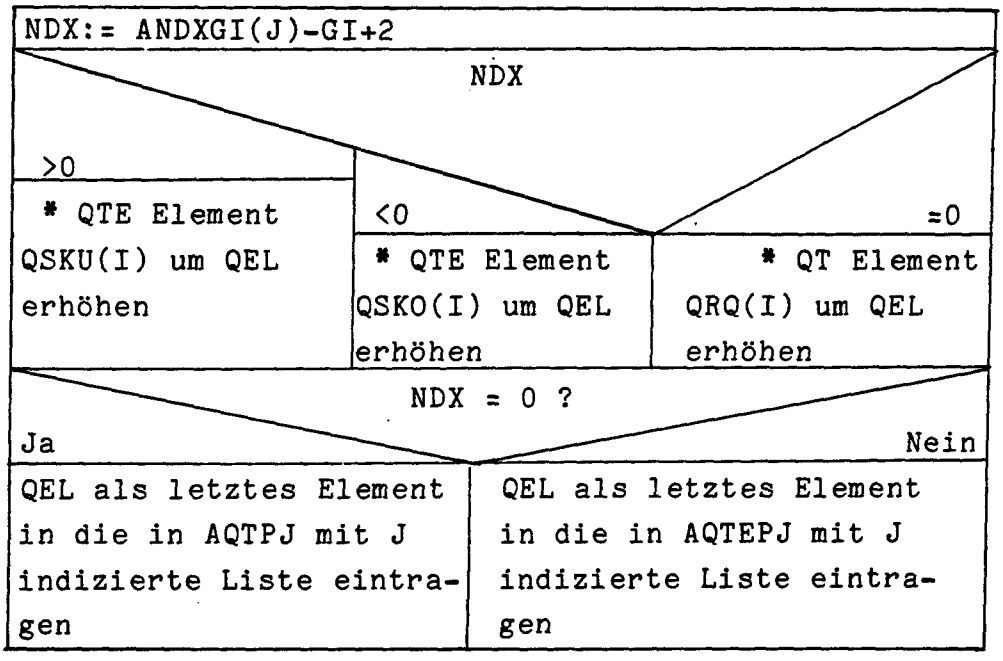
* Struktogramm



Initialisierung



Matrixelement QEL im Arbeitsbereich ablegen



5.3.3.6 ESTORE

* ESTORE - ENTRY von QSTORE, erledige Abschlußarbeiten nach Erzeugung aller Übergangsraten.

* Aufruf: CALL ESTORE

* Parameter

- Explizite Parameter

  Keine.

- Implizite Parameter

  - Eingabe

    ANZ INTEGER, COMMON AFZS. Anzahl n der Zustände des WS-Netzes.

    BNPLS1 INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.

    BN INTEGER*2, COMMON BFZS, Anzahl der Aufträge im WS-Netz.

    QRQ REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Spaltensumme aller Elemente der Blockdiagonaluntermatrizen von Q_π^T, bis auf die Diagonalelemente selbst.

    QSKO REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Spaltensummen der oberen Blocknebendiagonale.

QSKU REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Spaltensummen der unteren Blocknebendiagonale.

- Ausgabe

QRQ enthält nach Aufruf die Kehrwerte der Diagonalelemente von Q_{π}^{T}.

$FRMIN INTEGER*2, COMMON $LST, Minimalzahl der Freiplatzelemente in $LISTE während der Ausführung, Rückgabewert.

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
$LST

* Externe Routinen

- QSTQT: Übertrage eine Blockdiagonalmatrix von $LISTE nach QT.
- QSTQTE: Übertrage fertige Blocknebendiagonalmatrizen von $LISTE nach QTE.

* Funktionsablauf

Das bisherige Minimum an freien Listenelementen ist zu überprüfen und gegebenenfalls neu zu setzen.

Die Matrizen $Q^{T}_{BN+1,BN+1}$, $Q^{T}_{BN,BN-1}$, $Q^{T}_{BN,BN+1}$, $Q^{T}_{BN+1,BN}$ sind noch aus $LISTE und QLISTE nach QT bzw. QTE mit Hilfe von QSTQT bzw. QSTQTE zu übertragen.

Alle Spaltensummen werden in QRQ berechnet.

Abschließend erfolgt eine Überprüfung der Freiplatzliste auf Vollständigkeit.

* Besonderheiten

STOP 50: Arbeitsbereich $LISTE unvollständig wieder freigegeben.

STOP 51: Zu viele Freiplätze in der Freiplatzliste oder Ringverkettung.

STOP 52: Anfang der Freiplatzliste nicht auffindbar.

STOP 53: Ende der Freiplatzliste falsch markiert.

* Struktogramm

Für BNPLS1 wird QSTQT aufgerufen, für BN und BNPLS1 QSTQTE
Für NDX von 1 bis zur Zahl ANZ aller Feinzustände
QRQ(NDX):= -1.0 / (QRQ(NDX)+QSKO(NDX)+QSKU(NDX))

5.3.3.7 QSTQT

* QSTQT - Übertrage eine Blockdiagonaluntermatrix aus den Listen nach QT.

* Aufruf: CALL QSTQT(GIMNS1)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  GIMNS1 INTEGER, bezeichnet den Grobzustandsindex für die Blockdiagonaluntermatrix.

- Implizite Parameter

  - Eingabe

    API INTEGER, COMMON AFZS, Partition π in modifizierter Form.

    AQTPJ INTEGER, COMMON AFZS, Maximaldimension (2,AMAXPJ), aktuelle Dimension (2,ANZ). Enthält bei Aufruf Anfangs- und Endpunkte der Listen für alle Zeilen der in GIMNS1 bezeichneten Matrix.

    $LISTE INTEGER, COMMON $LST, Dimension (2,$LMAX). Enthält alle INTEGER-Elemente der in AQTPJ bezeichneten Listen.

    QLISTE REAL, COMMON QLST, Dimension $LMAX. Enthält alle REAL-Elemente der in AQTPJ bezeichneten Listen.

  - Ausgabe

    AQTPJ Enthält nach Aufruf pro Zeile den Index des ersten Elements der Zeile in QT und des zugehörigen Spaltenindex in AQT sowie die Anzahl der positiven Elemente der Zeile.

AQT INTEGER, COMMON AFZS, Maximaldimension AQTMAX, aktuelle Dimension AQTCUR-1. Enthält pro positivem Element einer Blockdiagonaluntermatrix dessen Spaltenindex. Diagonalelemente bleiben unberücksichtigt.

QT REAL, COMMON Q, Maximaldimension AQTMAX, aktuelle Dimension AQTCUR-1. Enthält alle positiven Elemente der Blockdiagonaluntermatrizen von Q_π^T (bis auf die Diagonalelemente selbst).

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
QLST
$LST

* Externe Routinen

- FREE: Stelle ein Listenelement in die Freiplatzliste.
- QALLOC: INTEGER FUNCTION, ordne ein freies Element aus den Matrizen QT und AQT zu.

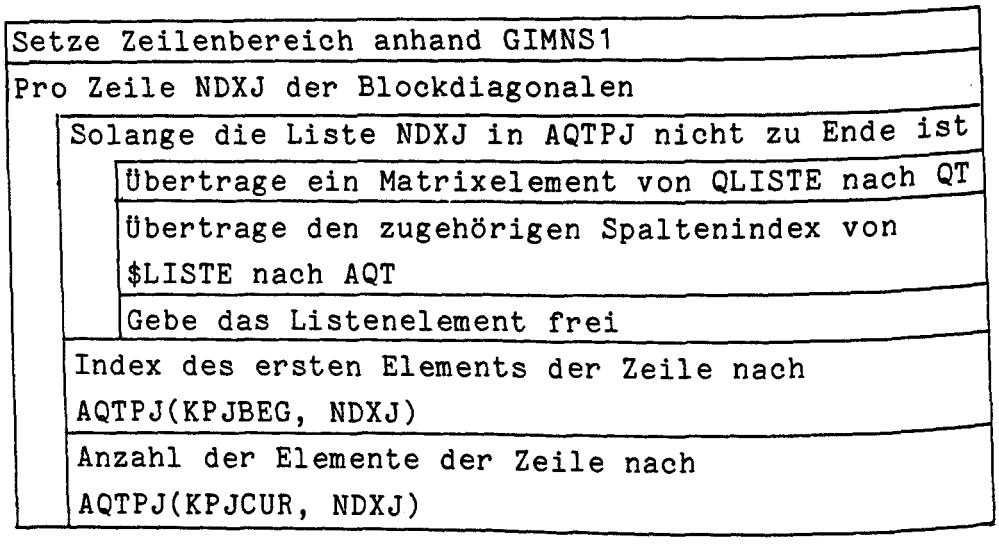
* Funktionsablauf

Für jede Zeile der in GIMNS1 bezeichneten Blockdiagonaluntermatrix sind die Elemente und deren Spaltenindizes nach QT und AQT zu übertragen. Die Anzahl dieser Elemente und der Index in QT und AQT des ersten sind in AQTPJ einzutragen, alle zugehörigen Listenelemente sind freizugeben.

* Besonderheiten

GIMNS1 wird nicht auf Gültigkeit überprüft.

* Struktogramm



5.3.3.8 QSTQTE

* QSTQTE - Übertrage Blocknebendiagonaluntermatrizen aus den Listen nach QTE.

* Aufruf: CALL QSTQTE(GIMNS2)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  GIMNS2 INTEGER, bezeichnet den Grobzustandsindex für die Zeilenindizes der Blocknebendiagonaluntermatrizen.

- Implizite Parameter

  - Eingabe

    API INTEGER, COMMON AFZS, Partition π in modifizierter Form.
    AQTEPJ INTEGER, COMMON AFZS, Maximaldimension (2,AMAXPJ), aktuelle Dimension (2,ANZ). Enthält bei Aufruf Anfangs- und Endpunkte der Listen für alle Zeilen der in GIMNS2 bezeichneten Matrizen.
    $LISTE INTEGER, COMMON $LST, Dimension (2,$LMAX). Enthält alle INTEGER-Elemente der in AQTEPJ bezeichneten Listen. QLISTE REAL, COMMON QLST, Dimension $LMAX. Enthält alle REAL-Elemente der in AQTEPJ bezeichneten Listen.

  - Ausgabe

    AQTEPJ Enthält nach Aufruf pro Zeile den Index des ersten Elements der Zeile in QTE und des zugehörigen Spaltenindex in AQTE sowie die Anzahl der positiven Elemente der Zeile.
    AQTE INTEGER, COMMON AFZS, Maximaldimension AQEMAX, aktuelle Dimension AQECUR-1. Enthält pro positivem Element der Blocknebendiagonaluntermatrizen dessen Spaltenindex.
    QTE REAL, COMMON Q, Maximaldimension AQEMAX, aktuelle Dimension AQECUR-1. Enthält alle positiven Elemente der Blocknebendiagonaluntermatrizen von Q_π^T.

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
QLST
$LST

* Externe Routinen

  - FREE: Stelle ein Listenelement in die Freiplatzliste.
  - EALLOC: INTEGER FUNCTION, ordne ein freies Element aus den Matrizen QTE und AQTE zu.

* Funktionsablauf

  Für jede Zeile der in GIMNS2 bezeichneten Blocknebendiagonaluntermatrizen sind die Elemente und deren Spaltenindizes nach QTE und AQTE zu übertragen. Die Anzahl dieser Elemente und der Index in QTE und AQTE des ersten sind in AQTEPJ einzutragen, alle zugehörigen Listenelemente sind freizugeben.

* Besonderheiten

  GIMNS2 wird nicht auf Gültigkeit überprüft.

* Struktogramm

Setze Zeilenbereich anhand GIMNS2	
Pro Zeile NDXJ der beiden Blocknebendiagonaluntermatrizen	
	Solange die Liste NDXJ in AQTEPJ nicht zu Ende ist
	Übertrage ein Matrixelement von QLISTE nach QTE
	Übertrage den zugehörigen Spaltenindex von $LISTE nach AQTE
	Gebe das Listenelement frei
	Index des ersten Elements der Zeile nach AQTEPJ(KPJBEG, NDXJ)
	Anzahl der Elemente der Zeile nach AQTEPJ(KPJCUR, NDXJ)

5.3.3.9 FREE

* FREE - Stelle ein Listenelement in die Freiplatzliste.

* Aufruf: CALL FREE(IEL)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  IEL INTEGER, Index des freizugebenden Listenelements.

- Implizite Parameter

  - Eingabe

    $FRBEG INTEGER*2, COMMON $LST, Index des ersten Elements der Freiplatzliste.
    $FREND INTEGER*2, COMMON $LST, Index des letzten Elements der Freiplatzliste.
    $FREE INTEGER*2, COMMON $LST, Anzahl der Elemente in der Freiplatzliste.
    $LISTE INTEGER, COMMON $LST, Dimension (2,$LMAX). Enthält die Elemente aller Listen.

  - Ausgabe

    $FRBEG Index des ersten Elements der Freiplatzliste.
    $FREND Index des letzten Elements der Freiplatzliste, gleich dem Wert von IEL.
    $FREE Neue Anzahl der Elemente der Freiplatzliste.
    $LISTE Enthält nach Aufruf an der Stelle $LISTE(KLSTNX,IEL) den Wert KNULL.

* Genutzte COMMON Bereiche

KONST (Codierhilfe, bleibt unverändert.)
$LST

* Externe Routinen

  Keine.

* Funktionsablauf

  Das mit IEL indizierte Element von $LISTE wird an das Ende der Freiplatzliste gebracht, die Anzahl der Freiplatzlistenelemente um 1 erhöht.

* Besonderheiten

  Keine.

* Struktogramm

Füge das Listenelement an das Ende der Freiplatzliste an
Erhöhe den Zähler der Freiplatzlistenelemente

5.3.3.10 ALLOC

* ALLOC - INTEGER FUNCTION, ordne den Index eines freien Listenelements zu.
* Aufruf: I=ALLOC(DUMMY)
* Parameter
  - Explizite Parameter

    DUMMY Beliebig, wird nicht referiert. Die Angabe von DUMMY ist trotzdem notwendig, weil in FORTRAN IV ALLOC sonst der Name einer Variablen ist.

- Implizite Parameter

  - Eingabe

    $FRBEG INTEGER*2, COMMON $LST, Index des ersten Elements der Freiplatzliste.
    $FREND INTEGER*2, COMMON $LST, Index des letzten Elements der Frei- platzliste.
    $FREE INTEGER*2, COMMON $LST, Anzahl der Elemente in der Freiplatzliste.
    $LISTE INTEGER, COMMON $LST, Dimension (2,$LMAX). Enthält die Elemente aller Listen.

  - Ausgabe

    $FRBEG Index des neuen ersten Elements der Freiplatzliste.
    $FREND Index des letzten Elements der Freiplatzliste.
    $FREE Neue Anzahl der Elemente der Freiplatzliste.

* Genutzte COMMON Bereiche

  KONST (Codierhilfe, bleibt unverändert.)
  $LST

* Externe Routinen

  Keine.

* Funktionsablauf

  Der Index des ersten Elements der Freiplatzliste ist Rückgabewert. Das nächste Element wird zum neuen ersten. Die Anzahl der Freiplatzlistenelemente wird um 1 vermindert.

* Besonderheiten

  Falls kein freies Element mehr vorhanden ist, STOP 7.

* Struktogramm

Freiplatzliste leer ?	
Ja	Nein
STOP 7	Verschiebe den Anfang der Freiplatzliste auf das bisherige zweite Element
	Gebe den Index des bisherigen ersten Elements an das rufende Programm

5.3.3.11 QALLOC

* QALLOC - INTEGER FUNCTION, ordne den Index des nächsten freien Elements in QT und AQT zu.

* Aufruf: I=QALLOC(DUMMY)

* Parameter

- Explizite Parameter

  DUMMY Beliebig, wird nicht referiert. Die Angabe von DUMMY ist trotzdem notwendig, weil in FORTRAN IV QALLOC sonst der Name einer Variablen ist.

- Implizite Parameter

  - Eingabe

    AQTMAX INTEGER, COMMON AFZS, Maximalzahl der Elemente in QT und AQT.
    AQTCUR INTEGER, COMMON AFZS, Index des ersten noch nicht belegten Elements in QT und AQT.

  - Ausgabe

    AQTCUR INTEGER, COMMON AFZS, Index des ersten noch nicht belegten Elements in QT und AQT.

* Genutzte COMMON Bereiche

AFZS

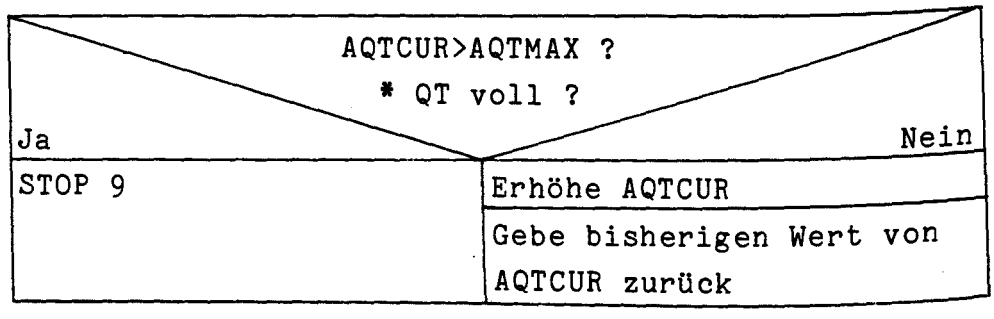
* Funktionsablauf

AQTCUR ist Rückgabewert und wird neu gesetzt.

* Besonderheiten

Falls kein Element in QT und AQT mehr frei ist, STOP 9.

* Struktogramm



5.3.3.12 EALLOC

* EALLOC - INTEGER FUNCTION, ordne den Index des nächsten freien Elements in QTE und AQTE zu.

* Aufruf: I=EALLOC(DUMMY)

* Parameter

- Explizite Parameter

  DUMMY Beliebig, wird nicht referiert. Die Angabe von DUMMY ist trotzdem notwendig, weil in FORTRAN IV EALLOC sonst der Name einer Variablen ist.

- Implizite Parameter

  - Eingabe

    AQEMAX INTEGER, COMMON AFZS, Maximalzahl der Elemente in QTE und AQTE.
    AQECUR INTEGER, COMMON AFZS, Index des ersten noch nicht belegten Elements in QTE und AQTE.

  - Ausgabe

    AQECUR INTEGER, COMMON AFZS, Index des ersten noch nicht belegten Elements in QTE und AQTE.

* Genutzte COMMON Bereiche

AFZS

* Funktionsablauf

AQECUR ist Rückgabewert und wird neu gesetzt.

* Besonderheiten

Falls kein Element in QTE und AQTE mehr frei ist, STOP 8.

* Struktogramm

AQECUR>AQEMAX ? * QTE voll ? Ja	Nein
STOP 8	Erhöhe AQECUR
	Gebe bisherigen Wert von AQECUR zurück

5.3.4 BGSA UND ABHÄNGIGE ROUTINEN

5.3.4.1 BGSA

* BGSA - Das Block-Gauß-Seidel Verfahren mit Aggregierung.

* Aufruf: CALL BGSA(EPS, NBGS, NABGS, DMAX)

* Parameter

- Explizite Parameter

  - Eingabe

    EPS REAL, Genauigkeitsschranke. Falls die maximale relative Abweichung zweier aufeinanderfolgender iterierter Lösungsvektoren kleiner als EPS ist, wird das Verfahren abgebrochen.
    NBGS INTEGER, Maximalzahl der durchzuführenden Itera- tionen.

  - Ausgabe

    NABGS INTEGER, Anzahl der durchgeführten Iterationen.
    DMAX REAL, maximale relative Abweichung zweier aufeinanderfolgender Lösungsvektoren bei Abbruch des Verfahrens.

- Implizite Parameter

  - Eingabe

    BNPLS1 INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.

* Genutzte COMMON Bereiche

BFZS

* Externe Routinen

- BGS: Führe einen Schritt des Block-Gauß-Seidel Verfahrens durch.
- GS: Ausführung eines Schrittes der Gauß-Seidel Punktiteration.
- AGG: Durchführung des Aggregierungsverfahrens.

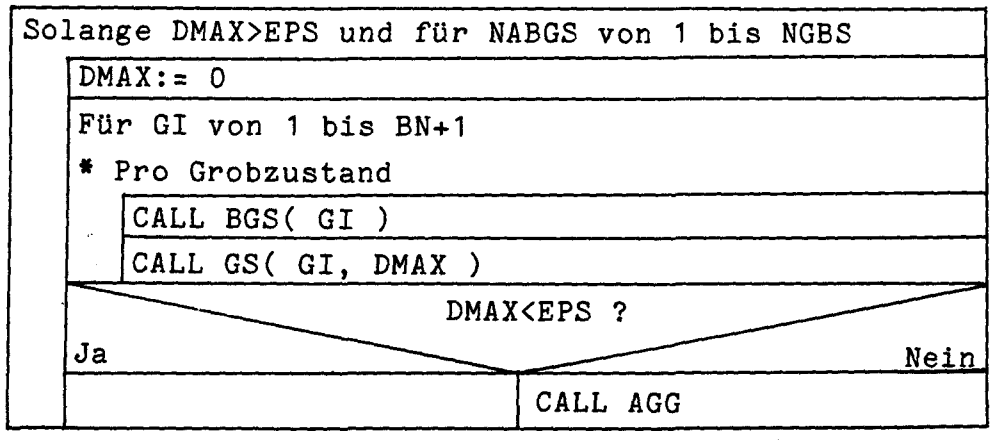
* Funktionsablauf

Solange die Maximalzahl der Iterationsschritte noch nicht erreicht worden ist, wird für jeden Grobzustand ein Schritt des BGS- und des GS- Verfahrens ausgeführt (CALL BGS, CALL GS).

Ist die im Laufe des GS Verfahrens festgestellte maximale relative Abweichung zweier iterierter Lösungsvektoren voneinander kleiner als der vorgegebene Wert EPS, so wird das Verfahren abgebrochen.

Sonst wird ein Aggregationsschritt (CALL AGG) ausgeführt und die Iteration anschließend fortgesetzt.

* Struktogramm



5.3.4.2 BGS

* BGS - Führe einen Schritt des Block-Gauß-Seidel Verfahrens durch.

* Aufruf: CALL BGS(GI)

* Parameter

- Explizite Parameter (nur Eingabe vorhanden)

  GI — INTEGER, bezüglich API transformierter Index des zu bearbeitenden Grobzustands.

- Implizite Parameter

  - Eingabe

    API — INTEGER, COMMON AFZS, Partition π in modifizierter Form.

    AQTEPJ — INTEGER, COMMON AFZS, Maximaldimension (2,AMAXPJ), aktuelle Dimension (2,ANZ).

    AQTE — INTEGER, COMMON AFZS, Maximaldimension AQEMAX, aktuelle Dimension AQECUR-1. Enthält pro positivem Element der Blocknebendiagonaluntermatrizen dessen Spaltenindex.

QTE REAL, COMMON Q, Maximaldimension AQEMAX, aktuelle Dimension AQECUR-1. Enthält alle positiven Elemente der Blocknebendiagonaluntermatrizen von Q_{π}^{T}.

QVN REAL, COMMON Q. Vektor der aktuellen Länge ANZ. Enthält die approximierte Grenzverteilung.

QBS REAL, COMMON Q. Enthält das einzige Nichtnullelement des Vektors $\tilde{b}$.

BNPLS1 INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.

- Ausgabe

QB REAL, COMMON Q. Arbeitsbereich für BGS und GS.

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
Q

* Externe Routinen

Keine.

* Funktionsablauf

Anhand des angegebenen Grobzustandsindex wird der Zeilenbereich der zu bearbeitenden Blockmatrizen gewählt.

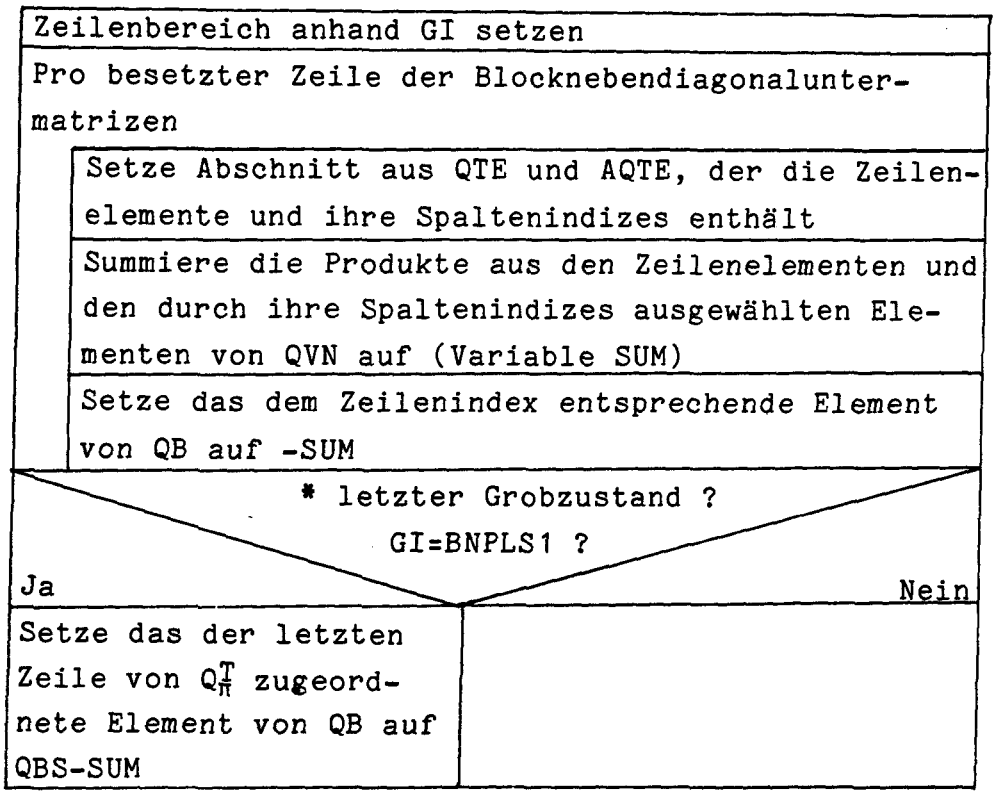
Das Produkt der Nebendiagonalmatrizen mit den durch AQTE angegebenen Komponenten von QVN wird zeilenweise berechnet. Der negative Wert davon wird den entsprechenden Komponenten von QB zugewiesen.

Für die letzte Zeile n des durch die Matrix Q_{π}^{T} beschriebenen Gleichungssystems existiert eine Komponente $\tilde{b}_n \neq 0$, QBS. Dort ist die Differenz aus QBS und dem genannten Produkt auf QB zuzuweisen.[9]

---

[9] Falls QBS genügend groß gewählt wird, ist das Ergebnis der Differenz gleich QBS.

* Struktogramm



5.3.4.3 GS

* GS - Ausführung eines Schritts der Gauß-Seidel Punktiteration.

* Aufruf: CALL GS(GI, DMAX)

* Parameter

- Explizite Parameter

  - Eingabe

    GI INTEGER, bezüglich API transformierter Index des zu bearbeitenden Grobzustands.

    DMAX REAL, Genauigkeitsschranke, bisheriger Maximalwert der relativen Abweichung zweier aufeinanderfolgender iterierter Lösungsvektoren.

  - Ausgabe

    DMAX neuer Maximalwert.

- Implizite Parameter

  - Eingabe

    API INTEGER, COMMON AFZS, Partition π in modifizierter Form.

    AQTPJ REAL, COMMON AFZS, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält pro Zeile den Index des ersten Elements dieser Zeile in QT und des zugehörigen Spaltenindex in AQT sowie die Anzahl der positiven Elemente der Zeile.

    AQT INTEGER, COMMON AFZS, Maximaldimension AQTMAX, aktuelle Dimension AQTCUR-1. Enthält pro positivem Element einer Blockdiagonaluntermatrix dessen Spaltenindex. Diagonalelemente bleiben unberücksichtigt.

    QT REAL, COMMON Q, Maximaldimension AQTMAX, aktuelle Dimension AQTCUR-1. Enthält alle positiven Elemente der Blockdiagonaluntermatrizen von Q_π^T (bis auf die Diagonalelemente selbst).

    QRQ REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Kehrwerte der Diagonalelemente von Q_π^T.

    QB REAL, COMMON Q. Arbeitsbereich für BGS und GS.

    QVN REAL, COMMON Q. Vektor der aktuellen Länge ANZ. Enthält die approximierte Grenzverteilung.

  - Ausgabe

    QVN Enthält die neue Approximation der Grenzverteilung, für den geforderten Grobzustand.

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST (Codierhilfe, bleibt unverändert.)
Q

* Externe Routinen

Keine.

* Funktionsablauf

Die durch GI spezifizierte Blockdiagonaluntermatrix wird bearbeitet.

Zeilenweise wird das Produkt dieser Matrix mit den durch AQT angegebenen Komponenten von QVN (ohne die Diagonalelemente) gebildet, das Ergebnis von der entsprechenden Komponente von QB subtrahiert, mit dem in QRQ enthaltenen Kehrwert multipliziert und neue Komponente von QVN.

Entsprechend der Größe der absolut genommenen relativen Abweichung dieser Komponente zur alten wird die Maximalabweichung bestimmt.

* Struktogramm

Zeilenbereich anhand GI setzen
Pro besetzter Zeile der Blockdiagonaluntermatrix
Setze Abschnitt aus QT und AQT, der die Zeilenelemente und ihre Spaltenindizes enthält
Summiere die Produkte aus den Zeilenelementen und den durch ihre Spaltenindizes ausgewählten Elementen von QVN auf (Variable SUM)
Berechne neues QVN(NDXJ)
Bestimme neue maximale relative Abweichung

5.3.4.4 AGG

* AGG - Durchführung des Aggregierungsverfahrens.

* Aufruf: CALL AGG

* Parameter

- Implizite Parameter

  - Eingabe

    QSKO REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Spaltensummen der oberen Blocknebendiagonale.

    QSKU REAL, COMMON Q, Maximaldimension APJMAX, aktuelle Dimension ANZ. Enthält die Spaltensummen der unteren Blocknebendiagonale.

    QVN REAL, COMMON Q. Vektor der aktuellen Länge ANZ. Enthält die approximierte Grenzverteilung.

    API INTEGER, COMMON AFZS, Partition π in modifizierter Form.

    BN INTEGER*2, COMMON BFZS, Anzahl der Aufträge im WS-Netz.

    BNPLS1 INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.

- Ausgabe

  QVN enthält die aufgrund der Aggregierung gewonnenen Werte.

* Genutzte COMMON Bereiche

  AFZS
  BFZS
  KONST (Codierhilfe, bleibt unverändert.)
  Q

* Externe Routinen

  - TRI: Löse lineares Gleichungssystem in Tridiagonalform.

* Funktionsablauf

  Pro durch API beschriebenen Zeilenbereich sind die aggregierten Koeffizienten zu bestimmen.

  Demzufolge wird pro Zeilenbereich das Skalarprodukt der entsprechenden oberen bzw. unteren Blocknebendiagonaluntermatrizen gebildet und in QG abgespeichert.[10] Das lineare Gleichungssystem in Tridiagonalform wird gelöst (CALL TRI) und die Lösung (QY) zur Gewinnung einer besseren Approximation der Gesamtgrenzverteilung QVN verwendet.

* Struktogramm

Für Grobzustand GI von 2 bis BNPLS1
GIMNS1:= GI-1
Setze Abschnitt aus QSKO und QVN zu Grobzustand GIMNS1
Setze Abschnitt aus QSKU und QVN zu Grobzustand GI
Bilde jeweils die Skalarprodukte und weise die Ergebnisse auf QG(KO,GIMNS1) bzw. QG(KU,GI) zu
CALL TRI(QG, QY)
* Löse aggregiertes Gleichungssystem
Pro Grobzustand GI
Multipliziere alle Elemente des zu GI gehörenden Teils von QVN mit QY(GI)

---

[10] Aufgrund des gewählten Lösungsverfahrens für Tridiagonalmatrizen ist eine Berechnung der Koeffizienten der Diagonale nicht notwendig.

5.3.4.5 TRI

* TRI - Berechne die Lösung eines linearen Gleichungssystems in Tridiagonalform.

* Aufruf: CALL TRI(QG, QY)

* Parameter

- Explizite Parameter

  - Eingabe

    QG REAL, Dimension (2,BMAXN), aktuell genutzt (2,BN). Enthält die Koeffizienten der oberen und unteren Nebendiagonalen des aggregierten Gleichungssystems.

  - Ausgabe

    QY REAL, Dimension BMAXN+1, aktuell genutzt BNPLS1. Enthält den Lösungsvektor des aggregierten Gleichungssystems.

- Implizite Parameter (nur Eingabe vorhanden)

  BNPLS1 INTEGER*2, COMMON BFZS, gleich BN+1. Anzahl der Grobzustände.

* Genutzte COMMON Bereiche

AFZS
BFZS
KONST

* Externe Routinen

Keine.

* Funktionsablauf

Es wird vorausgesetzt, daß $\tilde{b}_n$ und $\tilde{q}_{nn}$ so groß gewählt wurden, daß QVN(ANZ)= 1.0 .

Dann ist QY(BNPLS1)= 1.0, alle übrigen Werte ergeben sich direkt anhand der in der Verfahrensbeschreibung angegebenen Formeln.

* Struktogramm

QY(BNPLS1):= 1.0
Für NDXC von BN abwärts bis 1
QY(NDXC):= QY(NDXC+1) * QG(KO,NDXC) / QG(KU,NDXC)

6.0 TESTLÄUFE AUF DEM PIPELINERECHNER AP-190L

Einleitend werden die gegenüber der bisher beschriebenen seriellen Programmversion notwendigen Änderungen grob geschildert. Eine qualitative Beschreibung der Testläufe mit einer Diskussion der Ergebnisse schließt sich an.

Grundlage aller Testläufe war ein WS-Netz mit den in Beispiel 6.1.1 angegebenen Werten.

6.1 ÄNDERUNGEN

Die bei den Änderungen zu berücksichtigenden Einschränkungen aufgrund des verwendeten Rechners waren:

- kein eigenes Betriebssystem des AP
- geringer Programmspeicherplatz
- geringer Datenspeicherplatz
- kein Pipelining von INTEGER-Arithmetik
- gegenüber dem Host nur halb so große INTEGER Wortlänge

Demzufolge ergaben sich als Modifikationen:

- Kompilation und Binden auf dem Host. Datenübertragung, Laden und Starten vom Host aus.
- Zerlegung des Gesamtprogramms in für den Host und den AP geeignete Teile.
- Zerlegung der Datenbereiche und stückweise Übertragung von Host zu AP und zurück.
- Vermeidung der Auslagerung von Programmteilen mit starker Nutzung von INTEGER-Arithmetik, insbesondere komplizierterer Indexberechnung, auf den AP.
- Vor der Übertragung auf den AP sind Indexwerte, falls nötig, in den für den AP gültigen Bereich zu transformieren.

Für den AP ist keine direkte Rechenzeitmessung möglich. Alle Zeitmessungen geschahen daher vom Host aus und mußten demzufolge Realzeitmessungen sein. Aus diesem Grunde, und aus Gründen der Gesamteffektivität ist es notwendig, die Anzahl der Datentransfers zwischen Host und AP gering zu halten.

Zwei Grundversionen wurden getestet.

* Auslagerung des gesamten BGSA Verfahrens.

Notwendig war eine Aufspaltung der die Matrix Q repräsentierenden Variablen. Damit war eine Übertragung in die zugehörigen COMMON-Bereiche für den AP verbunden sowie eine Indextransformation für einige Vektoren (AQTPJ, AQTEPJ, AQT, AQTE).

Die für den Host geschriebenen FORTRAN Routinen wurden entsprechend angepaßt und zum Vergleich sowohl auf dem Host als auch auf dem AP (in der von APFTN erzeugten Form) ausgeführt.

Mit Beispiel 6.1.1 und N=10 konnte ein ganzer Iterationsschritt im AP ablaufen. Die Messung ergab, daß der Rechenzeitbedarf im Host schon dann etwa um das Dreifache anstieg. Bei N=15 mußte eine Zerlegung innerhalb jedes Iterationsschritts erfolgen. Dabei stieg der Zeitbedarf im Host erneut an.

* Auslagerung des Aggregierungsteils (AGG, TRI)

Die auf dem AP laufende Version wurde vierfach variiert, auf dem Host lief dagegen nur eine Version.

- Version HO

  Dies war eine reine FORTRAN Version für den Host. Die später auf den AP ausgelagerten Teile (Programme AGG und TRI) wurden in ihrem Ablauf so verändert, daß die auf dem AP genutzten Bibliotheksprogramme fast ohne sonstige Abwandlungen entsprechende Teilstücke der Version HO ersetzen konnten. Dadurch sollte ein direkter Vergleich dieser Ersetzung möglich gemacht werden.

- Version AF0

  Die Programme AGG und TRI wurden unverändert aus der Version HO übernommen und mit Hilfe des APFTN Compilers und APLOAD auf den AP gebracht. Diese Version enthielt keine Bibliotheksprogramme.

- Version AF1

  Gegenüber der Version AF0 wurden von APFTN erzeugte überflüssige Datentransfers eliminiert, sonst jedoch keine Änderungen vorgenommen.

- Version AB0

  Geeignete Teile der Programme AGG und TRI in Version AF0 wurden durch Bibliotheksroutinen ersetzt. In AGG wurde FVSMUL und FDOTPR verwendet. Die in TRI vorkommende rekursive Berechnung von QY ist eigentlich nicht zum Pipelining geeignet, insbesondere weil der AP die vorkommende Gleitkommadivision softwaremäßig vornehmen muß. Bei Verbleib von TRI auf dem Host wären aber noch mehr Datenübertragungen pro Verfahrensschritt notwendig geworden. Zur Beschleunigung gerade der Division wurde ausgenutzt, daß die Quotienten r_{i-1}/p_i noch ohne Rekursion berechnet werden können. Dafür stand die Bibliotheksroutine FVDIV zur Verfügung.

- Version AB1

  Hier wurden wieder von APFTN erzeugte überflüssige Datentransfers aus der Version AB0 eliminiert.

Aus der Tabelle 6.1.1 sind jeweils pro Version und einem typischen Beispiellauf Rechen- und Realzeitwerte zu entnehmen. Es muß jedoch berücksichtigt werden, daß bei den Realzeitmessungen für AGG sehr starke Schwankungen auftraten, z. B. lieferte die Version AB0 für N=18 und 124 durchgeführte Iterationen einmal den Wert 146.426 Sekunden.

Tabelle 6.1.1: Gemessener Zeitverbrauch

1. 15 Aufträge, Abbruch nach 100 Iterationen.

	Gesamt	BGSA	AGG
H0	15.305	8.934	3.164
AF0	13.063	6.791	58.546
AB0	13.043	6.748	38.056
AF1	12.904	6.656	27.880
AB1	12.954	6.647	42.368

2. 17 Aufträge, Abbruch nach 118 Iterationen.

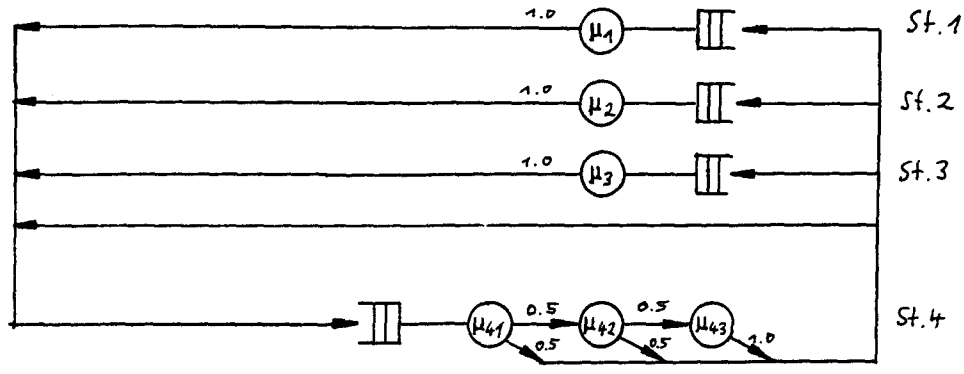
	Gesamt	BGSA	AGG
H0	22.935	13.812	8.853
AF0	20.257	11.135	38.804
AB0	20.348	11.206	52.131
AF1	20.063	11.002	82.477
AB1	20.114	10.974	29.255

3. 18 Aufträge, Abbruch nach 124 Iterationen.

	Gesamt	BGSA	AGG
H0	28.069	17.230	11.340
AF0	24.440	13.605	44.635
AB0	24.509	13.684	57.518
AF1	24.338	13.539	40.149
AB1	24.380	13.533	44.206

Die Spalten "Gesamt" und "BGSA" enthalten Rechenzeitwerte, die Spalte "AGG" Realzeitwerte. Die Differenz der Werte der AP-Versionen zur Host-Version in den Spalten "Gesamt" und "AGG" gibt ungefähr die Entlastung der Host-Zentraleinheit durch den AP wieder.

Beispiel 6.1.1 (nach (MUE 80)): Das verwendete WS-Netz



Die Zerlegung wurde für die in Tabelle 6.1.1 angegebenen Läufe an der Station 2 vorgenommen: k=2.

Die Stationen 1, 2 und 3 haben nur einen einphasigen Bediener. Die zugehörige Bedienzeitverteilung ist exponentiell, die möglichen internen Zustände sind 0 und 1, $l_1=l_2=l_3=1$.

Die Bedienraten sind konstant, $\mu^{(i)}(z)=\mu_i$, die Bedienstrategie ist FIFO.

Die Typmatrix für diese drei Stationen ist

$$T^{(i)}=\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

falls $n_i>0$.

Die Station 4 hat eine Bedienzeitverteilung vom Cox-Typ mit FIFO-Bedienstrategie, $l_4=3$. Die Bedienrate $\mu^{(4)}(z)$ ist abhängig vom internen Zustand c_4,

$\mu^{(4)}(1)=\mu_{41}$, $\mu^{(4)}(2)=\mu_{42}$, $\mu^{(4)}(3)=\mu_{43}$.

Die Ergebnisse in Tabelle 6.1.1 wurden mit folgenden Werten für μ erzielt:

$\mu_1=0.001$, $\mu_2=0.5$, $\mu_3=0.8$, $\mu_{41}=1.0$, $\mu_{42}=1.0$, $\mu_{43}=2.0$.

Die Typmatrix für $n_4>0$ ist

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 1 & 0 \end{matrix}$$

Die Wechselmatrix W lautet

$$\begin{matrix} 0.0 & 0.0 & 0.0 & 0.05 \\ 0.0 & 0.0 & 0.0 & 0.075 \\ 0.0 & 0.0 & 0.0 & 0.075 \\ 1.0 & 1.0 & 1.0 & 0.8 \end{matrix}$$

6.2 FOLGERUNGEN

Die Testläufe zeigen, daß zumindest der AP für das gewählte Problem nicht besonders geeignet ist:

- Für Probleme geringer Größenordnung und noch mittleren Speicherplatzbedarfs führte die notwendige Aufspaltung der Datenbereiche zu erhöhtem Rechenzeitaufwand im Host.

- Für Probleme mit einigem Rechenzeitaufwand, aber häufigem Transfer der berechneten Werte zwischen Host und AP sank zwar der Rechenzeitbedarf im Host, der Realzeitverbrauch stieg jedoch stark an. Der AP war dementsprechend mehr mit der Datenübertragung (und der Konvertierung der Gleitkommazahlen) beschäftigt als mit der eigentlichen Rechnung. Außerdem war es deswegen nicht möglich, die effektive Rechenzeit auf dem AP festzustellen.

Eine Möglichkeit, die Realzeitmessungen zuverlässiger zu machen, würde darin bestehen, die Verweildauer der Daten im AP "künstlich" zu erhöhen, indem man das Aggregierungsverfahren für ein- und dieselben Werte entsprechend häufig ausführt und so den Anteil der Rechenzeit an der gemessenen Realzeit ausreichend erhöht.

Für die Lösung des gewählten numerischen Problems wäre ein Pipelinerechner, der auch Indexberechnungen effektiv und für Zahlen ausreichender Größenordnung durchführen kann, wegen der vorliegenden dünnbesetzten Matrix erfolgversprechender.

Im Verlauf der Tests zeigte sich außerdem, daß der Rechenzeitaufwand zur Erstellung der Matrix in ungefähr der gleichen Größenordnung wie der für die eigentliche Berechnung liegt. Die zur Matrixaufstellung benutzten Algorithmen bedienen sich jedoch, die eigentliche Ratenberechnung ausgenommen, nur der INTEGER Arithmetik, sind demnach schon deshalb nicht für den AP geeignet. Auch die Form der Algorithmen (z. B. FZ, GNTRNS) scheint nicht für eine Bevorzugung eines Pipelinerechners zu sprechen.

Falls im vorliegenden Fall aber für ein- und dieselbe Struktur des WS-Netzes, d. h. bei Erhaltung der Besetzungsstruktur der Matrix, nur für unterschiedliche Parameterwerte Lösungen gesucht sind, ist es günstig - wie auch in (MUE 80) getan -, die Erzeugung der Matrixstruktur von der Berechnung zu trennen. Die dafür geeignete Stelle liegt in der Routine GNRATE.

SYMBOLE, ABKÜRZUNGEN

$:=$	definitionsgemäß gleich
$\underline{\ }$	Vektorschreibweise, z. B. $\underline{x}$
$\lvert\ \rvert$	Absolutbetrag oder Mächtigkeit einer Menge
$\lVert\ \rVert_1$	Summennorm eines Vektors
Δt	Zeitintervall
μ	Menge der Funktionen zur Bedienratenbestimmung
$\mu(i)$	Die bedienratenfestlegende Funktion der Station i
π	Partition von Z(S)
π_I	I-te Komponente der Partition π
Ω	Raum der Elementarereignisse
ω	Ein Elementarereignis aus Ω
AP	bezeichnet den Pipelinerechner AP-190L
B	Anzahl der Bediener, die eine Station umfaßt
$\underline{B}$	Inhomogenität eines aggregierten Gleichungssystems
c_i	Interner Zustand der Station i
E	Einheitsmatrix
exp	Exponentialfunktion
F	Wahrscheinlichkeitsverteilung von X(t)
f	Dichtefunktion einer Zufallsvariablen X(t)
G	Koeffizientenmatrix eines aggregierten Gleichungssystems
G_I	Grobzustand I
I	Index bezüglich der Grobzustände
J(z)	Jacksonzustand mit dem Repräsentanten z
k	Station, an der ein WS-Modell S zerlegt wird
l_i	Anzahl der internen Zustände der Station i, ohne den Ruhezustand
M	Anzahl der Stationen in einem WS-Netz
m	Anzahl von Zufallsvariablen
N	Anzahl der Aufträge im WS-Netz
NCD	Abkürzung für "nearly completely decomposable"
$\mathbb{N}$	Menge der natürlichen Zahlen einschließlich der 0
$\mathbb{N}_1$	Die Menge der natürlichen Zahlen von 0 bis 1
n	Anzahl der Zustände eines Zustandsraums
n_i	Anzahl der Aufträge an Station i
n(I)	Anzahl der Feinzustände von G_I
P	Wahrscheinlichkeitsmaß über Ω
p_{ij}	Übergangswahrscheinlichkeit für zeitlich homogene Markovketten
$\hat{p}_{ij}$	Übergangswahrscheinlichkeit für Markovketten
$\dot{p}_{ij}$	Ableitung der Übergangswahrscheinlichkeit p_{ij}

Q	Matrix der Übergangsraten
Q_π	Matrix der Übergangsraten, in der Zeilen und Spalten nach der Partition π geordnet sind
q_{ij}	Übergangsrate, Element der Matrix Q bzw. Q_π
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}_0^+$	Menge der reellen, nichtnegativen Zahlen
S	WS-Modell der Modellklasse WSM
$Stat(S)$	Menge der Stationen des WS-Modells S
S_1	Teilsystem von S
S_2	Teilsystem von S
T	Menge der Typen aller Stationen
$T(i)$	Typ der Station i
t	Zeit (Index)
t_i	Zeit (Index)
$\underline{v}$	Gleichgewichtsvektor eines WS-Netzes
W	Wechselmatrix
WS	Abkürzung für: Warteschlange(n)
WSM	Modellklasse von WS-Netzwerkmodellen
w_{ji}	Element der Wechselmatrix W
$X(t)$	Zufallsvariable zum Index t
x	Wert aus dem Bildbereich von X(t)
$Z(S)$	Menge der Zustände des WS-Modells S
z	Zustand aus dem Zustandsraum Z(S)

STOP CODES

STOP	Routine	Bedeutung
0	MSG	Eingabefehler und DIALOG=.FALSE.
1	SETPI	API zu niedrig dimensioniert (kann nur auftreten, falls die Dimensionierung nicht anhand BMAXM vorgenommen wurde).
2	PUSHK	Funktion PUSH für vollen Stack ($STCKK) verlangt.
	POPK	Funktion POP für leeren Stack ($STCKK) verlangt.
3	PUSHZ	Funktion PUSH für vollen Stack ($Z) verlangt.
	BACKZ	Entfernen des letzten Eintrags aus leerem Stack ($Z) verlangt.
4	SETZS	BZS zu niedrig dimensioniert (Schätzwert ändern).
5	SETJZ	AJZ zu niedrig dimensioniert. Schätzwert ändern.
6	GNRATE	Aufruftyp unbekannt (Fehler in der Parameterübergabe).
7	ALLOC	Anfang der Freiplatzliste nicht auffindbar, bzw. keine freien Listenelemente mehr.
8	EALLOC	AQE zu niedrig dimensioniert. Schätzwert (AQEMAX) ändern.
9	QALLOC	AQT zu niedrig dimensioniert. Schätzwert (AQTMAX) ändern.
10	QSTQE	Spaltenindex aus falschem Grobzustandsbereich.
12	MSG	Ungültige Nachrichtennummer (MSGNO).
13	WSNET	QB zu niedrig dimensioniert. Schätzwert ändern.
20	SRCHZS	Spezifizierter Feinzustand im angegebenen Indexbereich nicht auffindbar.
50	ESTORE	Arbeitsbereich $LISTE unvollständig wieder freigegeben.
51		Zu viele Freiplätze in der Freiplatzliste oder Ringverkettung.
52		Anfang der Freiplatzliste nicht auffindbar.
53		Ende der Freiplatzliste falsch markiert.
54	QSTORE	Zeilenindex nicht aus gültigem Grobzustandsbereich (Ursache z. B. falsche Aufruffolge von QSTORE).

LITERATURVERZEICHNIS

(BAU 78) Bauer, Heinz: Wahrscheinlichkeitstheorie und Grundzüge der Maßtheorie,

Berlin, 1978.

(BLU 72) Blum, E. K.: Numerical Analysis And Computation. Theory And Practice,

Reading, Mass., 1972.

(BOA 82) Bolch, G., Akyildiz, I. F.: Analyse von Rechensystemen,

Stuttgart, 1982.

(CHU 78) Chung, K. L.: Elementare Wahrscheinlichkeitstheorie und stochastische Prozesse,

Berlin, 1978.

(COU 77) Courtois, P. J.: Decomposability. Queuing and Computer System Applications,

New York, 1977.

(FHW 79) Fritz, F.-J., Huppert, B., Willems, W.: Stochastische Matrizen,

Berlin, 1979.

(FPS 78) Floating Point Systems, Inc. (Hrsg.): Programmers Reference Manual,

Part One, Beaverton, Oregon, 1978.

(FPS 80) Floating Point Systems, Inc. (Hrsg.): AP Math Library,

Vol. 2, Beaverton, Oregon, 1980.

(KLE 75) Kleinrock, Leonard: Queuing Systems, Vol. I, Queuing Theory,

o. O., 1975.

(MUE 80) Müller, Bruno: Zerlegungsorientierte, numerische Verfahren für Markovsche Rechensystemmodelle, Dissertation,

Dortmund, 1980.

(RAL 77) Ramamoorthy, C. V., Li, H. F.: Pipeline Architecture,

ACM Computing Surveys, Vol. 9, No. 1, März 1977, S. 61 - 102.

(SIA 61) Simon, H. A., Ando, A.: Aggregation of variables in dynamic systems,

Econometrica 29, S. 111 - 138. Nachgedruckt in: Ando, A., Fisher, F. M., Simon, H. A.: Essays on the Structure of Social Science Models,

S. 64 - 91, MIT Press, Cambridge, Massachusetts, 1963.

(SOM 83) Sommer, A.: Scheduling-Strategien für Pipeline-Rechner,

Spezielle Berichte der KFA Jülich, Nr. 190, Januar 1983.

(STE 78) Stewart, William J.: A Comparison of Numerical Techniques in Markov Modeling,

Communications of the ACM, Vol. 21, No. 2, Feb. 1978.